universität wien

DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

„Einführung in die Programmierung mit Visual Basic for Applications mit Microsoft Excel“

verfasst von / submitted by

Christina Fel

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magistra der Naturwissenschaften (Mag.rer.nat.)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 190 406 884
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Lehramtsstudium: UF Mathematik / UF Informatik und Informatikmanagement
Betreut von / Supervisor:	ao. Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Wien, am 8. Mai 2018

Christina Fel

Danksagung

Mein erster Dank geht an meine Diplomarbeitsbetreuerin ao. Univ.-Prof. Dipl. -Ing Dr. Renate Motschnig, welche mich mit ihrem stets konstruktiven und professionellen Feedback bei der Erstellung dieser Diplomarbeit sehr unterstützt und gelenkt hat.

Danke sagen möchte ich aber auch an die Lehrerinnen und Lehrer, welche ihre Zeit geopfert haben und mein Lehrerinnen- und Lehrhandbuch im Rahmen des „Proof of Concepts“ verifiziert haben und mir wichtige Anregungen und Tipps gaben.

Ich möchte aber auch meinen Studienkolleginnen und Studienkollegen Danke sagen. Danke für die viele Unterstützung und auch die Geduld, welche vielleicht manchmal von Nöten war. Insbesondere geht mein diesbezüglicher Dank an meine Freundin Birgit Pindhofer, sie war während meines gesamten Studiums eine wichtige Stütze für mich, wir haben vieles gemeinsam durchgestanden und absolviert.

Mein besonderer Dank gilt aber meiner Familie, welche mich stets während meines Studiums unterstütz hat und dieses überhaupt erst ermöglichte. Mir wurde geholfen, wenn ich Hilfe brauchte, ich wurde motiviert, wenn ich Ansporn brauchte, und ich wurde getröstet, wenn ich Zuspruch und Trost brauchte.

Abstract

Deutsch

Die gegenständliche Diplomarbeit beschäftigt sich mit den Möglichkeiten der Wissensvermittlung in der Programmierung in der „Neuen Oberstufe“ (Nost) an allgemeinbildenden höheren Schulen (AHS). Diese Wissensvermittlung soll im Rahmen der schulautonomen Möglichkeiten unter Verwendung von Visual Basic for Applications (VBA) erfolgen.

Im theoretischen Teil dieser Diplomarbeit werden Basisinformationen zu Visual Basic for Applications gegeben. Weiters beschäftigt sich dieser Teil der Diplomarbeit mit dem Lehrplan der „Neuen Oberstufe“ (Nost) mit dem Ziel, einen Zusammenhang der schulautonomen Möglichkeiten und der Wissensvermittlung in der Programmierung durch Visual Basic for Applications mit Microsoft Excel herzustellen. Anschließend werden verschiedene Lerntheorien und didaktische Ansätze beziehungsweise Unterrichtsmethoden näher untersucht. Dies erfolgt mit dem Ziel deren Anwendbarkeit bei der Umsetzung des im praktischen Teil dieser Diplomarbeit zu erstellenden Handbuches für Lehrerinnen und Lehrer für den Einstieg in die Programmierung mit Visual Basic for Applications zu prüfen.

Im praktischen Teil dieser Diplomarbeit wird ein Lehrrinnen und Lehrerhandbuch für den Einstieg in die Programmierung mit Visual Basic for Applications erstellt. Es werden darin Erklärungen, Materialen und Musteraufgaben für den Erwerb und die Vermittlung entsprechenden Basiswissens ausgearbeitet und zur Verfügung gestellt. Das „Proof of Concept“ des Handbuches erfolgt über eine Begutachtung durch Informatiklehrerinnen und Informatiklehrer. Die Ergebnisse daraus werden analysiert, reflektiert und schlussendlich in das Handbuch eingearbeitet.

English

This diploma thesis deals with the possibilities of knowledge transfer in programming in the „Neuen Oberstufe" (Nost) at the “allgemeinbildenden höheren Schulen” -academic secondary school-upper cycle. This transfer of knowledge should take place in the framework of the school-autonomous possibilities by using Visual Basic for Applications.

The theoretical part of this diploma thesis provides basic information about Visual Basic for Applications.

Furthermore it deals with the curriculum of the „Neue Oberstufe“ (Nost) with the aim to establish a connection between the school-autonomous possibilities and the knowledge transfer in programming with Visual Basic for Applications with Microsoft Excel.

It subsequently examines various learning theories and didactic teaching methods in more detail. These insights are necessary for the practical part of this thesis which includes a handbook for teachers detailing the first steps of programming with Visual Basic for Applications with Microsoft Excel.

The handbook contains explanations and sample tasks for the acquisition and conveyance of basic knowledge. The proof of concept of the handbook is carried out by a survey with a questionnaire for teachers. The results will be analysed, reflected and finally incorporated into the handbook.

Inhalt

I. Einleitung

I.1 Motivation

Die Motivation, mich in meiner Diplomarbeit mit dem Thema „Einführung in die Programmierung mit Visual Basic for Applications mit Microsoft Excel“ zu beschäftigen begründet sich darin, dass ich selbst eine „Modulare Oberstufe“ einer allgemein höherbildenden Schule (Schulversuch MOST) besucht habe. Dieser Schulversuch (MOST) stellt den Vorläufer der „Neuen Oberstufe“ (NOST) dar. Ich genoss dabei zwar eine fundierte Allgemeinausbildung mit dem Schwerpunkt auf Sprachen, erwarb aber keinerlei Kenntnisse im Schreiben von Computerprogrammen, dem sogenannten „Coding“. Diese fehlenden Kenntnisse und Fähigkeiten stellten mich und viele meiner Studienkolleginnen und Studienkollegen in der Anfangsphase unseres Informatikstudiums vor nicht unerhebliche Probleme und große Herausforderungen.

Die flächendeckende Umsetzung der „Neuen Oberstufe“ erfolgt an mindestens 3-jährigen Oberstufenformen ab der 10. Schulstufe (6. Klasse an allgemeinbildenden höheren Schulen und II. Jahrgang beziehungsweise 2. Klasse an berufsbildenden mittleren und höheren Schulen (BMHS), land- und forstwirtschaftlichen Schulen beziehungsweise der Bundesbildungsanstalt für Elementarpädagogik (BAfEP) / Bildungsanstalt für Sozialpädagogik (BASOP)).

Die Schulen können sodann im schulautonomen Bereich auch ergänzende Schwerpunkte, unter anderem auch im Bereich der Informations- und Kommunikationstechnologie, setzen. Dies ist im aktuellen Lehrplan der allgemeinbildenden höheren Schulen so verankert.

Mit dieser Stärkung der Schulautonomie besteht die Möglichkeit den Schülerinnen und Schülern im regulären Schulunterricht, im Rahmen von Wahlpflichtgegenständen, entsprechende „Coding“-Kenntnisse zu vermitteln.

I.2 Forschungsfrage

Meine Diplomarbeit beschäftigt sich mit der Frage, welche Lerntheorien, Unterrichtsmethoden beziehungsweise didaktische Ansätze besonders gut dazu geeignet

sind, um den Einstieg in die Programmierung mit Visual Basic for Applications (VBA) mit Microsoft Excel an Oberstufen allgemeinbildender höherer Schulen (AHS) im Rahmen schulautonomen Möglichkeiten der „Neuen Oberstufe“ zu unterrichten?

Weiters sollen auch die Vorteile beziehungsweise die Nachteile bei der Verwendung von Visual Basic for Applications (VBA) mit Microsoft Excel beschrieben werden.

I.3 Struktur der Arbeit

Der einleitende Theorieteil beschäftigt sich mit unterschiedlichen Lerntheorien und didaktischen Ansätzen. Diese sind die Basis für das erstellte Lehrerinnen- und Lehrerhandbuch zur Vermittlung von einführendem Wissen im „Coding“, im speziellen in die Einführung der Programmierung mit VBA in Microsoft Excel.

Im theoretischen Teil ist ein für den Schulunterricht relevanter Grobüberblick über die Entwicklung der VBA-Programmierung mit Excel enthalten. Ebenso sind die bemerkenswertesten Vor- und Nachteile der Programmierung in Visual Basic for Applications (VBA) mit Excel aufgelistet und erläutert.

Im praktischen Teil befindet sich die Ausarbeitung von Materialien, welche für den Erwerb der erforderlichen Kenntnisse für den Einstieg in die VBA-Programmierung mit Excel erforderlich sind. Neben der Erläuterung der Bedienung des Entwicklertools in Microsoft Excel, welches den Visual Basic for Applications (VBA)-Teil inkludiert, werden die wichtigsten Techniken für den Einstieg in die Programmierung zusammengefasst. Es werden Musterprogramme zur Veranschaulichung der Lerninhalte entwickelt und zur Verfügung gestellt. Dieser praktische Teil der Diplomarbeit ist in der Form eines Lehrerinnen- und Lehrerhandbuches gestaltet, welches das erforderliche Wissen für die Abhaltung einer Lehrveranstaltung im Rahmen der schulautonomen Möglichkeiten der „Neuen Oberstufe“ bündelt.

Ich möchte den Schülerinnen und Schülern einen Anstoß dazu geben, die Liebe zur Informatik, im Rahmen einer Lehrveranstaltung des Unterrichtes der „Neuen Oberstufe“, in sich selbst zu entdecken und zu entwickeln.

I.4 Erwartete Ergebnisse

Als Ziel meiner Diplomarbeit sollen in der Praxis brauchbare und anwendbare didaktische Ansätze sowie ein verwendbares Lehrerinnen- und Lehrerhandbuch zur Wissensvermittlung für den Einstieg in das Programmieren in der „Neuen Oberstufe“, unter Verwendung von Visual Basic for Applications (VBA) mit Microsoft Excel, entstehen. Das Lehrerinnen- und Lehrerhandbuch soll es den Lehrerinnen und Lehrern ermöglichen, ohne viel Zusatzaufwand, eine entsprechende Lehrveranstaltung im Rahmen der schulautonomen Möglichkeiten der „Neuen Oberstufe“ in einer allgemeinbildenden höheren Schule (AHS) abzuhalten und den Schülerinnen und Schülern einführende Kenntnisse im „Coding“ zu vermitteln.

Das „Proof of Concept“ meiner Ausarbeitungen erfolgte im Rahmen einer Begutachtung durch erfahrene Lehrerinnen und Lehrer. Es wird ein Fragebogen entworfen und konstruktives Feedback und weiterführende Anregungen, nach meiner persönlichen Reflexion, in die finale Version des Lehrerinnen- und Lehrerhandbuches, welches einen Bestandteil meiner Diplomarbeit bildet, eingearbeitet beziehungsweise aufgenommen.

I.5 Konnex zur Informatik

Insbesondere der praktische Teil der Diplomarbeit zeigt auf, in welchem Ausmaß und Umfang sich meine Diplomarbeit mit Themen der Informatik auseinandersetzt. Es werden Fragen aufgeworfen, analysiert und gelöst. Gerade die Problemanalyse und eine folgende Problemlösung im Rahmen der bestehenden technischen und persönlichen Möglichkeiten stellt meiner Meinung nach eine der wichtigsten Fähigkeiten von Informatikerinnen und Informatikern dar.

II. Theoretischer Teil

II.1 Visual Basic for Applications (VBA)

II.1.1 Was ist Visual Basic for Applications (VBA)?

Visual Basic for Applications (VBA) ist eine Programmiersprache, welche, wie der Name schon verrät, nur innerhalb einer anderen Applikation ausgeführt werden kann, in diesem Fall eben in Microsoft Excel. Visual Basic for Applications (VBA) stellt somit eine in Microsoft Excel integrierte Programmiersprache dar. (vgl. Buhl & Strauch , 2005, S. 2)

Es handelt sich dabei um die sonst autonom funktionierende Programmiersprache Visual Basic von Microsoft, welche um spezielle Ergänzungen für die verschiedenen Anwendungen von Microsoft Office erweitert wurde. (vgl. Theis , 2016, S. 17) Trotz der bestehenden Ähnlichkeiten zu den eigenständigen Produkten „Visual Basic 6“ oder dem Nachfolgeprogramm „Visual Basic .NET“ sind diese nicht immer kompatibel. Insbesondere gegenüber Visual Nasic.NET gibt es viele Änderungen. (vgl. Kofler & Nebelo, 2016, S. 7)

Es hat sich eine Makrosprache aus dem Wunsch heraus entwickelt, um eigene Funktionen erstellen zu können und um wiederholt auftretende Befehle einer automatisierten Abarbeitung zuzuführen. Weiters wurde eine makrogesteuerte Veränderung von Menüs und die Erstellung eigener Dialoge ermöglicht. Bis zur Version 4 von Microsoft Excel hat sich eine sehr unübersichtliche Makrosprache entwickelt, wodurch eine fast uneingeschränkte Programmierung aller Excel-Funktionen möglich war. Viele Probleme konnten aber nur sehr umständlich gelöst werden, was zu fehlerhaften und langsamen Programmen führte. Weiters erschwerend kam hinzu, dass andere Microsoft Office Anwendungen (zum Beispiel Word oder Access) eine eigene Makrosprache hatten. Aus diesem Grund entschloss sich Microsoft eine eigene Makroprogrammiersprache, welche auf der Programmiersprache Visual Basic von Microsoft basiert, zu entwickeln, welche schließlich mit der Version 5 von Microsoft Excel eingeführt wurde. Anfangs stand diese neue Makroprogrammiersprache nur für Microsoft Excel zur Verfügung, mittlerweile ist diese aber in allen Microsoft-Office-Anwendungen integriert. Während ursprünglich noch ein deutscher VBA-Dialekt verfügbar war, werden seit der Excel Version 97 nur mehr englische Schlüsselwörter verwendet. Die Einsatzmöglichkeiten von Visual Basic for Applications (VBA) sind mittlerweile so

umfangreich, dass damit völlig eigenständige Programme entwickelt werden können, welchen kaum mehr anzumerken ist, dass es sich dabei um eine Excel-Anwendung handelt. (vgl. Kofler & Nebelo, 2016, S. 3ff.)

II.1.2 Merkmale von Visual Basic for Applications (VBA)

Visual Basic for Applications ist eine vollwertige Programmiersprache, welche zum Beispiel alle in „echten" Programmiersprachen wie C++ oder Java üblichen Variablentypen unterstützt. Visual Basic for Applications (VBA) kann ebenso mit Zeichenketten umgehen, dynamische Felder verwalten und zur Erstellung rekursiver Funktionen verwendet werden. (vgl. Kofler & Nebelo, 2016, S. 6)

II.1.2.1 VBA ist objektorientiert

Unter Objekten versteht man beispielsweise Arbeitsmappen, Tabellen oder Zellen sowie Zellenbereiche. Diese besitzen Merkmale, wie zum Beispiel die Ausrichtung oder die Farbe, welche über die Eigenschaften verändert werden können. Unter „Eigenschaften" versteht man also vordefinierte Schlüsselwörter, welche zur Veränderung (Manipulation) von Objekten verwendet werden. Zusätzlich gibt es noch „Methoden", welche mit herkömmlichen Kommandos verglichen werden können. Es können damit komplexe Operationen, wie zum Beispiel das Erzeugen von Objekten (zum Beispiel Diagramme, Tabellen), durchgeführt werden. „Methoden" lassen sich aber nur auf dafür vorgesehene Objekte anwenden. (vgl. Kofler & Nebelo, 2016, S. 6)

II.1.2.2 VBA ist ereignisorientiert

Beim Eintreten bestimmter Ereignisse, wie zum Beispiel dem Anklicken eines Buttons oder der Aktivierung einer Tabelle, werden damit verknüpfte Makros (Programme) automatisch ausgeführt. Die Tätigkeit der Programmiererin beziehungsweise des Programmierers beschränkt sich also auf das Schreiben der Ereignisprogramme, diese müssen sich aber nicht um die Verwaltung dieser Ereignisse kümmern. Diese Aufgabe wird von Excel erledigt. (vgl. Kofler & Nebelo, 2016, S. 7)

II.1.2.3 VBA ist erweiterungsfähig

Mit der Verwendung von Visual Basic for Applications (VBA) kann auch auf Objekte anderer Anwendungen zugegriffen werden. Das bedeutet, dass zum Beispiel in einem Excel-Programm auch die Schlüsselwörter, der so genannten Objektkatalog, anderer Programme, wie beispielsweise Access oder Word, genutzt werden können. Es können so mittels Add-Ins neue Objekte oder Excelfunktionen erstellt werden. Mit dem integrierten Dialog-Editor kann die Interaktion zwischen Computer und Nutzer gesteuert werden. Dieser Dialog-Editor ist ebenfalls objekt- und ereignisorientiert. (vgl. Kofler & Nebelo, 2016, S. 7)

II.1.2.4 VBA ermöglicht eine effiziente Fehlersuche

Zur professionellen Fehlersuche können Programmteile Schritt für Schritt ausgeführt und Variableninhalte überwacht werden. Bei Eintritt bestimmter vordefinierter Bedingungen kann die Programmausführung unterbrochen werden. (vgl. Kofler & Nebelo, 2016, S. 7)

II.1.3 Vorteile und Nachteile von Visual Basic for Applications (VBA) für den Einstieg in die Programmierung in Schulen

Im Allgemeinen können nachstehende Vorteile bzw. Nachteile von Visual Basic for Applications (VBA) genannt werden:

- Aufgaben werden in Excel immer auf die gleiche Weise ausgeführt, wobei diese Konsistenz meistens von Vorteil ist.
- Mittels Excel können viele Aufgaben viel schneller erledigt werden als manuell.
- Wurde ein Makro fehlerfrei erstellt, so wird dieses immer korrekt ausgeführt. Natürlich resultiert daraus auch ein Nachteil, wenn Makros fehlerbehaftet erstellt wurden.
- Mit Hilfe zur Verfügung gestellter Makros können auch mit Excel unerfahrene Personen aufgaben korrekt erledigen.
- Langwierige und zeitraubende Berechnungen und Bearbeitungen können automatisiert und somit vereinfacht werden.
- Die VBA-Programmierung muss gelernt werden, was einige Zeit in Anspruch nimmt.

- Ein nicht unerheblicher Nachteil ist, dass Excel regelmäßig mit Upgrades von Microsoft versorgt wird. Es ist dabei nicht unbedingt gewährleistet, dass bestehende Makros immer funktionieren und fehlerfrei bleiben, gegebenenfalls werden Anpassungen erforderlich.

- Als Nachteil von VBA sei erwähnt, dass keine selbstständigen Programme kompiliert werden können. Bei der Nutzung der erstellten Programme muss der Computer des Anwenders immer über eine vorhandene Excel-Installation verfügen.

(vgl. Walkenbach, 2016, S. 36f.)

Neben den vorstehenden allgemeinen Vorteilen und Nachteilen von Visual Basic for Applications (VBA), welche ich bestätigen kann, konnte ich im Rahmen meines Studiums und der Beschäftigung mit der Materie im Zusammenhang mit der Erstellung meiner Diplomarbeit noch zusätzliche Positiva und Negativa erkennen:

- Es ist neben der Installation von Microsoft Office keine weitere Softwareinstallation erforderlich. VBA steht mit der bloßen Installation von Microsoft Excel zur Verfügung. Es fallen keine weiteren Lizenzkosten an.

- VBA erfüllt viele Anforderungen, welche heute an moderne Programmiersprachen gestellt werden. VBA ist im Aufbau, der Syntax und der Arbeitsweise mit anderen Programmiersprachen vergleichbar. Nach dem Erwerb von Grundkenntnissen in der Programmierung mit VBA sollte somit ein Umstieg in eine andere moderne Programmiersprache, wie zum Beispiel C+ oder Java, leichter fallen.

- Neben dem Erwerb von Wissen in der Programmierung werden auch Kenntnisse über die Interaktion zwischen dem eigenen Programmcode und Microsoft Excel vermittelt. Gerade dadurch erscheint der im Lehrplan für die Neue Oberstufe zusätzliche Aspekt für die Nutzung der schulautonomen Möglichkeiten gegeben.

- Die Microsoft Office Anwendungen und somit auch Microsoft Excel sind in der Privatwirtschaft und im öffentlichen Bereich sehr verbreitet. Jeglicher Erwerb von Zusatzkenntnissen im Zusammenhang mit diesen Anwendungen sollten also für ein folgendes Studium oder auch für eine Jobsuche von Vorteil sein.

- Es steht eine umfangreiche Online-Hilfefunktion zur Verfügung, welche von Microsoft betrieben und gewartet wird.

II.2 Verankerung des Themas im Lehrplan der „Neuen Oberstufe“

II.2.1 Was ist die „Neue Oberstufe“ (Nost)?

Die neue Oberstufe soll als neues modernes pädagogisches Gesamtkonzept für eine Verstärkung der Individualisierung und Kompetenzorientierung beitragen. Ziel ist die Steigerung der Motivation bei den Schülerinnen und Schülern, sowie eine Erhöhung der Erfolgsquoten. (vgl. BMBMF Bundesministerium für Bildung, Wissenschaft und Forschung, 2018)

II.2.2 Welche Schultypen sind betroffen?

Eingeführt wird die neue Oberstufe an mindestens 3-jährigen Oberstufenformen ab der 10. Schulstufe, das bedeutet ab der 6. Klasse an allgemeinbildenden höheren Schulen (AHS), dem II. Jahrgang beziehungsweise der 2. Klasse an berufsbildenden mittleren und höheren Schulen (BMHS), land- und forstwirtschaftlichen Schulen, an Bundesbildungsanstalten für Elementarpädagogik (BAfEP) beziehungsweise den Bundesbildungsanstalten für Sozialpädagogik und Elementarpädagogik (BASOP). (vgl. BMBMF Bundesministerium für Bildung, Wissenschaft und Forschung, 2018)

II.2.3 Zeitplan für die Umsetzung der flächendeckenden Einführung der NOST

Die neue Oberstufe wurde an wenigen Standorten ab dem Schuljahr 2013/2014 versuchsweise begonnen. (vgl. BMB Bundesministerium für Bildung, 2016)

Die flächendeckende Umsetzung der Einführung der neuen Oberstufe war für das Schuljahr 2017/2018 vorgesehen wobei die Möglichkeit bestand, dass schulautonom der Start auf die Schuljahre 2018/2019 beziehungsweise 2019/2020 verlegt werden kann. (vgl. BMBMF Bundesministerium für Bildung, Wissenschaft und Forschung, 2018)

Zuletzt wurde die Frist für die Umsetzung der Einführung der neuen Oberstufe an Schulen um zwei Jahre nach hinten verschoben, sodass diese nunmehr ab dem Schuljahr 2021/2022 verpflichtend umzusetzen ist. Diese Verlängerung soll dazu genutzt werden um die Oberstufenreform zu evaluieren sowie um den Lehrerinnen und Lehrern, aber auch den

Schülerinnen und Schülern, Zeit zu geben, sich auf dieses neue System vorzubereiten. (vgl. Die Presse, 2018)

II.2.4 Ziele der neuen Oberstufe

Die Einführung der neuen Oberstufe verfolgt zwei Hauptziele. Diese sind einerseits die Verankerung bedarfsgerechter Fördermaßnahmen mit einhergehender individueller Lernbegleitung für Schülerinnen und Schüler und andererseits die semesterweise Lehrstoffverteilung in Kompetenzmodulen. Es soll dadurch schrittweise die kontinuierliche Leistungserbringung der Schülerinnen und Schüler gefördert werden und diese auf ein Hochschulstudium oder/und das Erwerbsleben besser vorbereiten. (vgl. BMB Bundesministerium für Bildung, 2016)

Im Einzelnen sind folgende Zielsetzungen mit der Einführung der neuen Oberstufe verbunden: (vgl. BMBMF Bundesministerium für Bildung, Wissenschaft und Forschung, 2018)

- Da sowohl im Wintersemester als auch im Sommersemester in allen Gegenständen positive Leistungen erbracht werden müssen, kommt es zu einer „Verdichtung der Lernaktivitäten“. Ein „Schleifenlassen“ und „Nachlernen über Ostern“, um das Schuljahr zu schaffen, ist nicht mehr möglich.

- Die Schülerinnen und Schüler sind stärker für den eigenen Lernerfolg verantwortlich, wodurch deren „Eigenverantwortung“ gestärkt wird. Im Bedarfsfall steht den Schülerinnen und Schülern aber eine individuelle Lernbegleitung zur Verfügung, bei welcher ein persönlich abgestimmtes Unterstützungsangebot in Anspruch genommen werden kann.

- Durch die „Semestergliederung“ werden die Schülerinnen und Schüler lernorganisatorisch besser auf einen Hochschulrhythmus vorbereitet, da dies an Hochschulen gleich ist.

- Es kommt zu einer Optimierung des „Frühwarnsystems“. Die neue individuelle Lernbegleitung wird mit dem Fachunterricht und dem Förderunterricht in einem neuen pädagogischen Rahmen in Beziehung gesetzt.

- „Negativ beurteilte beziehungsweise nicht beurteilte Pflichtgegenstände“ in einem Semester können durch Semesterprüfungen nachträglich positiv korrigiert werden, wobei grundsätzlich diese zweimal wiederholt werden können.
- „Schulstufenwiederholungen“ werden dadurch reduziert, weil Schülerinnen und Schüler eine Schulstufe nur dann wiederholen müssen, wenn diese in einem Semesterzeugnis zwei „Nicht genügend“ oder zwei Nichtbeurteilungen haben beziehungsweise, wenn diese in der Oberstufe bereits einmal mit drei „Nicht genügend“ oder drei Nichtbeurteilungen in die nächste Schulstufe aufgestiegen sind.
- Schülerinnen und Schüler mit besonderen „Begabungen“ profitieren von individuellen Fördermöglichkeiten. Es können auch Schulstufen übersprungen und Teilprüfungen der Reife- beziehungsweise Reifediplomprüfung vorgezogen werden.

II.2.5 Der Lehrplan der allgemeinbildenden höheren Schule

Im ersten Teil „Allgemeines Bildungsziel“ des Lehrplanes für allgemein höherbildende Schulen ist festgehalten, dass der Lehrplan einerseits die für die Einheitlichkeit und Durchlässigkeit des Schulwesens notwendigen Vorgaben darstellt. Andererseits soll dieser Freiräume eröffnen, die der Konkretisierung am jeweiligen Schulstandort vorbehalten sind. Er ist somit unter anderem auch die Grundlage für das „standortbezogene Bildungsangebot“. Die Stundentafeln geben das vorgesehene Stundenausmaß in den jeweiligen Unterrichtsfächern an und definieren auch den Freiraum für schulautonome Maßnahmen. Es ist in den „Leitvorstellungen“ genannt, dass innovative Technologien bezüglich Information und Kommunikation, sowie der Massenmedien immer stärker in alle Lebensbereiche eindringt und digitale Kompetenz zu fördern ist. Es ist auch angeführt, dass im Sinne einer gemeinsamen Bildungsentwicklung aller Unterrichtsgegenstände eine fächerübergreifende Vernetzung und Ergänzung zu berücksichtigen ist. Im Punkt „Bildungsbereiche“ sind unter anderem auch die Bereiche „Medienerziehung“ und „Erziehung zur Anwendung neuer Technologien“ angeführt. Im Bildungsbereich „Natur und Technik“ ist genannt, dass für die Problemanalyse und -lösung Kompetenzen der Formalisierung, Modellbildung, Abstraktions- und Raumvorstellungsvermögen zu vermitteln sind. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

Im zweiten Teil „Allgemeine didaktische Grundsätze“ des Lehrplanes für allgemein höherbildende Schulen ist angeführt, dass unter anderem der Unterricht an Vorkenntnisse der Schülerinnen und Schüler anzuknüpfen hat. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

Im dritten Teil „Schul- und Unterrichtsplanung“ wird unter anderem auch bestimmt, dass besondere Formen der Ausstattung konstruktiv in die Unterrichtsarbeit einzubringen sind, wobei die Konkretisierung und Realisierung der Vorgaben § 17 des Schulunterrichtsgesetzes standortbezogen zu erfolgen hat. Diese Planungen beziehen sich neben weiteren auch auf die schulautonomen Lehrplanbestimmungen. Im Rahmen der schulautonomen Möglichkeiten können ergänzende Schwerpunkte im Ausmaß von jeweils acht Wochenstunden in den Bereichen Fremdsprachen, Mathematik, Naturwissenschaften, Informations- und Kommunikationstechnologie, Ökologie, Wirtschaft, Musisch-kreativer Schwerpunkt und Sport gesetzt werden. Verwiesen wird dabei auf den sechsten Teil des Lehrplanes. Es ist festzulegen, welche Bildungs- und Lehraufgaben, sowie Lehrstoffbereiche über die dort festgelegten Inhalte hinausgehen. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

Der vierte Teil des Lehrplanes sieht vor, dass in Gymnasien mit der Ermächtigung für schulautonome Lehrplanbestimmungen in der Oberstufe mindestens zwei Stunden Informatik zu unterrichten sind. Bei Gymnasien ohne die Ermächtigung für schulautonome Lehrplanbestimmungen sind in der 5. Klasse (9. Schulstufe) genau zwei Wochenstunden Informatik zu unterrichten. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

Der sechste Teil des Lehrplanes gibt den Lehrstoff vor und gliedert sich in vier Bereiche. Bei den Pflichtgegenständen in der 5. Klasse (1. u. 2. Semester) ist die Informatik und deren Bezug auf die Menschen und die Gesellschaft zu behandeln. Weiteres sind Informatiksysteme im Hinblick auf den Aufbau der digitalen Endgeräte, deren Funktionsweisen, das Betriebssystem und die Vernetzung unterrichtsgegenständlich. Die „Angewandte Informatik“ beschäftigt sich mit Standardsoftware, zum Beispiel für Kalkulationen (zum Beispiel Microsoft Excel). In der praktischen Informatik geht es um Begriffe und Konzepte der Informatik, die Grundprinzipien von Automaten, Algorithmen und Datenstrukturen, sowie um Datenbanken und einfache Datenmodelle. Algorithmen

sollen erklärt, entworfen, dargestellt und in einer Programmiersprache implementiert werden können. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

Die Wahlpflichtgegenstände der 6. bis 8. Klassen schließen an das Pflichtfach der 5. Klasse an. Die Bildungs- und Lehraufgabe sowie der Lehrstoff gliedern sich analog zur 5. Klasse in die Inhaltsdimensionen „Informatik, Mensch und Gesellschaft“, „Informatiksysteme“, „Angewandte Informatik“ und „Praktische Informatik“. Die Handlungsdimensionen gliedern sich dabei in die Bereiche „Wissen und Verstehen“, „Anwenden und Gestalten“ sowie „Reflektieren und Bewerten“. Ausdrücklich ist angeführt, dass es den Lehrerinnen und Lehrern in Abstimmung mit den Schülerinnen und Schülern obliegt in bestimmten Bereichen Schwerpunkte zu setzen oder Bereiche exemplarisch zu behandeln. Im Bereich der angewandten Informatik ist in den jeweiligen Kompetenzmodulen der 6. bis 8. Klasse angeführt, dass aufbauend Algorithmen zu entwerfen, formal darzustellen, zu verstehen, zu implementieren, in Bezug auf deren Effizienz zu bewerten und zu testen sind. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

II.2.6 Zusammenhang von Lehrplan, den schulautonomen Möglichkeiten der „Nost“ und Visual Basic for Applications mit Microsoft Excel

Im nachstehenden Teil wird der Zusammenhang zwischen dem Lehrplan und den schulautonomen Möglichkeiten der neuen Oberstufe in Verbindung mit der Wissensvermittlung von Programmierkenntnissen in Visual Basic for Applications mit Microsoft Excel aufgezeigt.

Bezugnehmend auf die vorangeführten Auszüge aus dem Lehrplan für allgemeinbildende höhere Schulen und der nachstehenden ergänzenden Erläuterungen ist festzuhalten, dass das Unterrichten von Programmierkenntnissen, also auch der Einstieg in die Programmierung mit Visual Basic for Applications mit Microsoft Excel, im Rahmen der schulautonomen Möglichkeiten der neuen Oberstufe (Nost) möglich ist.

Im Hinblick auf den ersten Teil des Lehrplanes begründet sich dies damit, dass dadurch digitale Kompetenzen vermittelt werden und die Schülerinnen und Schüler zur Anwendung neuer Technologien erzogen werden. Zusätzlich erfolgt einhergehend mit der Vermittlung von Programmierkenntnissen auch die Kompetenzvermittlung in Bezug auf die

Formalisierung, Modellbildung und Abstraktion. Bezüglich des zweiten Teiles des Lehrplanes wird darauf verwiesen, dass in der 9. Schulstufe, im Rahmen des Unterrichtes, bereits Microsoft Office Kenntnisse und somit auch Microsoft Excel-Kenntnisse vermittelt werden. Für das Programmieren mit Visual Basic for Applications mit Microsoft Excel sind diese bereits vermittelten Kenntnisse sehr von Vorteil. In Bezug auf den dritten Teil des Lehrplanes kann darauf verwiesen werden, dass die bestehende Ausstattung konstruktiv in den Unterricht einbezogen wird, da keine weitere Hardware oder Software anzuschaffen ist. Es ist dabei die Ausstattung, welche bereits in der 9. Schulstufe verwendet wird, vollkommen ausreichend. Auch das Ausmaß der zur Verfügung stehenden Unterrichtsstunden ist für die Wissensvermittlung ausreichend, wobei sich auch eine fächerübergreifende Unterrichtsplanung, zum Beispiel mit Mathematik oder Physik, anbietet. Bezugnehmend auf die vorstehenden Ausführungen zum sechsten Teil des Lehrplanes ist festzuhalten, dass bei einem entsprechenden Angebot an Wahlpflichtfächern der 6. bis 8. Klasse im Unterrichtsfach Informatik die Wissensvermittlung von Programmierkenntnissen von Visual Basic for Applications (VBA) mit Excel vollinhaltlich im Lehrplan gedeckt ist. Dies begründet sich damit, dass mit Visual Basic for Applications (VBA) komplexe Algorithmen programmiert werden können. Voraussetzung dafür ist, dass diese entworfen, modelliert, verstanden und implementiert werden können. Als Testumgebung für die entwickelten Programme in Visual Basic for Applications (VBA) steht ein Direktausgabebereich sowie ein „Debugger“, also ein Werkzeug zur Diagnostik und Fehlersuche in Computersystemen zur Verfügung.

Im Rahmen einer Schwerpunktsetzung ist in den schulautonomen Lehrplanbestimmungen festzulegen, welche Bildungs- und Lehraufgaben sowie Lehrstoffbereiche der schulautonomen Lehrpläne über den sechsten Teil (Lehrpläne der einzelnen Unterrichtsgegenstände) festgelegten Inhalte hinausgehen. (vgl. Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018)

Hinsichtlich der Lehrstoffbereiche ist diesbezüglich anzuführen, dass neben dem Umgang mit einer modernen objekt- und ereignisorientierten Programmiersprache wie „Visual Basic for Applications“, wie im Lehrplan verankert, darüberhinausgehend auch Wissen über die Interaktion der Programme mit den Microsoft-Office-Anwendungen, hier insbesondere Microsoft Excel, vermittelt werden. Da die Microsoft-Office-Anwendungen sowohl in der Privatwirtschaft als auch im öffentlichen Bereich sehr verbreitet sind, werden die

Schülerinnen und Schüler nicht nur im Sinne der Bildungs- und Lehraufgaben des Lehrplanes unterrichtet. Die Schülerinnen und Schüler erlangen darüberhinausgehend ein erweitertes Spektrum bezüglich der Möglichkeiten der Verwendung und Automatisierung der Microsoft-Office-Anwendungen, welche bei einer allfälligen Jobsuche und im späteren Berufsleben sehr von Vorteil sein können.

II.3 Lerntheorien

II.3.1 Vorwort

Im Rahmen dieser Diplomarbeit und in Anbetracht auf die Beantwortung meiner Forschungsfrage wurde eine umfangreiche Literaturrecherche durchgeführt.

Im folgenden Teil werden die wichtigsten Lerntheorien erläutert und es wird herausgearbeitet, welche Lerntheorien bei der Umsetzung des Lehrerinnen- und Lehrerhandbuches besonders geeignet erscheinen.

II.3.2 Beschreibung

Unter Lerntheorien versteht man in der Psychologie Theorien zur systematischen Erklärung von nicht beobachtbaren Lernprozessen. Die bedeutungsvollsten Lerntheorien sind dabei das klassische Konditionieren, das operante Konditionieren sowie die sozial-kognitive Lerntheorie und das Lernen durch Einsicht. (vgl . Stangl, 2018 zit. nach Edelmann, 1993)

In der Pädagogik werden die menschlichen Lernprozesse, die drei klassischen Lerntheorien Behaviorismus, Kognitivismus und Konstruktivismus, unterschieden. (vgl . Stangl, 2018 zit. nach Edelmann, 1993)

II.3.3 Behavioristische Lerntheorien

Bei diesen Lerntheorien werden die Lernenden als Individuen aufgefasst, welche durch bestimme Reizkonstellationen und Verhaltenskonsequenzen in ihrer Verhaltensweise oder Reaktion gelenkt werden können. (vgl. Tulodziecki, Herzig, & Blömeke, 2017, S. 55)

Man spricht dabei von Lernen, wenn sich eine beobachtbare Reaktion auf einen Reiz ändert, eben vergrößert oder verkleinert. So spricht man zum Beispiel vom Lernen, wenn ein Schüler sich verstärkt mit Beiträgen in den Unterricht einbringt (beobachtbares Verhalten), weil er dafür vorher gute Beurteilungen bekommen hat (Konsequenz). Die Vorgänge im Gehirn bleiben dabei völlig unbeachtet, weshalb die oder der Lernende als „black box“ bezeichnet wird. (vgl. Tulodziecki, Herzig, & Blömeke, 2017, S. 35f.)

Das Lernen findet durch eine Reiz-Reaktionskette statt und ist beendet, sobald diese Reiz-Reaktionskette hergestellt wurde. Bei diesen Modellen wird der Lernerfolg durch

Verstärkung oder Abschwächung von Verhaltensweisen verstanden. Es gibt dabei eine positive Verstärkung durch Belohnung und eine negative Verstärkung (Abschwächung) durch Bestrafung (fehlende Belohnung). Der oder die Lernende verhält sich dabei von innen heraus passiv und regiert auf äußere Reize aktiv mit einer Reaktion. Dem oder der Lehrenden kommt eine zentrale Rolle zu und setzt geeignete Anreize. Auf die Reaktionen der Schülerinnen und Schüler gibt er oder sie entsprechende Rückmeldungen. Durch diese positiven oder negativen Rückmeldungen greifen die Lehrenden zentral in den Lernprozess der Lernenden ein. (vgl. Meir, 2006, S. 10f.)

II.3.4 Kognitivistische Lerntheorien

Bei diesen Lerntheorien werden die Lernenden als Individuen verstanden, welche über Interaktionen mit deren Umwelt Informationen aufnehmen und verarbeiten. Dabei wird Wissen erworben, welches in der Folge nachgeahmt und übertragen werden kann. (vgl. Tulodziecki, Herzig, & Blömeke, 2017, S. 55)

Man spricht dann von Lernen, wenn die Lernenden sich Wissen über ein bestimmtes Thema aneignen, welches diese vorher nicht hatten. Dies ist zum Beispiel dann der Fall, wenn diese nach der Erarbeitung des Themas „Erster Weltkrieg“ eine Vorstellung darüber ausbilden, warum dieser Konflikt entstand. Es werden dabei neben kognitiven Strukturen auch die Prozesse (Wahrnehmung, Einordnung, Verankerung, Reproduktion und Transfer) für die Wissensveränderung betrachtet. Äußere Reize und Informationen werden basierend auf dem Vorwissen und der Erfahrung verarbeitet. Dieser Ansatz stößt an seine Grenzen, wenn bei anstehenden Problemlösungen nicht auf bereits bestehende Wissensreproduktionen beziehungsweise Wissenstransfers zurückgegriffen werden kann. (vgl. Tulodziecki, Herzig, & Blömeke, 2017, S. 36f.)

Lernen bezieht sich auf die Annahme von Informationen sowie auf deren Verarbeitung und Speicherung. Im Hauptfokus steht dabei der Verarbeitungsprozess in Verbindung mit den passenden und unterstützenden Methoden sowie Problemstellungen. Von höchster Bedeutung in Bezug auf die Beeinflussung des Lernprozesses sind dabei das Lernangebot selbst, die Informationsaufbereitung und die Problemstellung sowie die Methodik. Durch die Lösung von Problemen erwerben die Lernenden neue Kenntnisse welche deren Wissen vergrößern. Die Lernenden nehmen dabei eine aktive Rolle ein, indem diese selbst

Informationen aufnehmen, verarbeiten und Probleme lösen. Die Lernenden nehmen eine bedeutende Rolle im Lernprozess ein. Den Lehrenden kommt hinsichtlich der didaktischen Aufbereitung der Aufgabenstellungen eine zentrale Rolle zu. Diese stellen Informationen zur Verfügung geben die Problemstellung vor. Ihnen kommt eine unterstützende Funktion bei der Bearbeitung der Informationen durch die Lernenden zu. (vgl. Meir, 2006, S. 12f.)

II.3.5 Konstruktivistische Lerntheorien

Bei diesen Lerntheorien nehmen die Lernenden Informationen nicht nur durch die Interaktion mit der Umwelt auf, sondern nehmen die Situationen, in welchen sich diese befinden, subjektiv wahr und gestalten diese durch deren eigenes Handeln mit. Es wird dabei situationsbezogenes Wissen durch einen selbstorganisierten Prozess konstruiert. Diese subjektiven Konstruktionen von Erkenntnissen werden durch die Kommunikation mit anderen Menschen zu sozial relevanten Wirklichkeitskonstruktionen. (vgl. Tulodziecki, Herzig, & Blömeke, 2017, S. 55f.)

Diese Ansätze eignen sich dazu um in Lernprozessen die Fähigkeit zu entwickeln um selbstgesteuert Probleme lösen zu können. Die Schwächen liegen im strukturierten Wissensaufbau. (vgl. Tulodziecki, Herzig, & Blömeke, 2017, S. 38)

Der Lernprozess wird als individuelle Wissenskonstruktion angesehen. Daraus leitet sich ab, dass es kein richtiges oder falsches Wissen gibt. Es bestehen lediglich divergierende Sichtweisen, welche aus der individuellen Erfahrungswelt der Lernenden hervorgehen. Das Ziel besteht darin, dass die Lernenden fähig sind, Lösungen aus Situationen heraus selbstständig zu entwickeln. Die Lernenden nehmen eine zentrale Rolle ein. Diese müssen aus wenigen Vorgaben, basierend aus vorhandenen Kompetenzen und Wissen, selbstorientiert passende Lösungen finden. Den Lehrenden kommt neben den Aufgaben der Wissensvermittlung und Problemstellung auch die Rolle eines Lernbegleiters zu, um die Lernenden bei deren sozialen Lernprozessen zu unterstützen. (vgl. Meir, 2006, S. 14f.)

II.3.6 Reflexion der Lerntheorien in Bezug auf eine mögliche Umsetzung des Lehrerinnen- und Lehrerhandbuches

Wie komplex der Vorgang des Lernens ist, kann schon alleine aus der Unterschiedlichkeit der vorher genannten Lerntheorien abgeleitet werden. Gerade diese Komplexität macht es

sehr schwierig eine Lerntheorie vor dem Hintergrund auszuwählen, die doch sehr anspruchsvolle Materie der Programmierung Schülerinnen und Schülern näher zu bringen. Viel mehr erscheint die Verwendung eines Mix der Lerntheorien zielführend zu sein. Eine Kombination aus fremdgesteuerten und selbstorientierten Lernsequenzen ermöglicht sowohl die effektive Vermittlung von Basiswissen als auch eine problemorientierte Umsetzung der erworbenen Kenntnisse.

Durch die Anwendung von auf behavioristischen Lerntheorien basierenden Lernmethoden kann lehrerzentriet sehr effektiv das umfangreiche erforderliche Basiswissen beziehungsweise Faktenwissen vermittelt werden, welches eine unbedingte Voraussetzung für den weiteren Lernprozess darstellt. Dieses Grundlagenwissen besteht aus den möglichen Programmierbefehlen und deren Syntax. Das Lernziel dieser Phase ist das Merken und Wiedererkennen des vom Lehrer dargebotenen Stoffes.

Für die erste Wissensvermittlung bezüglich der Arbeitsweise der Computerbefehle bietet sich die Verwendung von Unterrichtsmethoden an, welche auf kognitivistischen Lerntheorien basieren. Die Schülerinnen und Schüler lernen am Modell durch Einsicht die mögliche Anwendung der zur Verfügung stehenden Befehle, es wird somit das nötige Konzeptwissen vermittelt. Das Lernziel dieser Phase ist, dass entsprechendes Verständnis bezüglich des Stoffes aufgebaut wird.

Für die praxisorientierte Umsetzung des zuvor durch fremdgesteuertes Lernen erworbenen Wissens bieten sich auf konstruktivistische Lerntheorien basierende Unterrichtsmethoden an. Mit Hilfe des selbstorganisierten Lernens durch persönliche Erfahrung wird den Schülerinnen und Schülern eine individuelle Problemlösung abverlangt, dies bei einer gleichzeitigen sozial relevanten Wirklichkeitskonstruktion. Anhand dieser Kompetenz kann schlussendlich die Lehrkraft feststellen ob die nötigen Kompetenzen erworben wurden und somit die Wissensvermittlung funktioniert hat. Das Ziel dieser Phase ist, dass die Schülerinnen und Schüler kreativ mit dem erworbenen Wissen umgehen und dieses reflektieren können.

Unter Zugrundelegung der zuvor genannten Erkenntnisse werden in der Folge Unterrichtsmethoden behandelt, welche auf den vorhergenannten Lerntheorien aufbauen.

II.4 Didaktische Ansätze/Unterrichtsmethoden

II.4.1 Was versteht man unter Didaktik im Allgemeinen?

Unter Didaktik versteht man die Wissenschaft vom Lehren und Lernen, wobei alle Aspekte im komplexen Zusammenhang von Entscheidungen, Begründungen, Voraussetzungen und Prozessen für den Unterricht umfasst sind. Aus didaktischer Sicht lassen sich Lehr- und Lernprozesse in zwei zentrale Bereiche aufteilen und bestehen aus einer Zieldimension und einer Wegdimension. Die Zieldimension umfasst die Ziele und die Inhalte und stellt die Frage nach dem „Was?“ und „Wozu?“. Die Wegdimension umfasst die Methoden und die Medien, wobei es um das „Wie?“ und „Womit?“ geht. (vgl. Riedl , 2004, S. 8)

Der Begriff Didaktik im engeren Sinn umfasst nur die Zieldimension, also die Bildungsintention und die Bildungsinhalte. Die Methodik, also die Methoden und die Medien, besteht als eigenständiger Bereich. Diese Trennung ist heute nicht mehr von Bedeutung und man versteht jetzt unter dem Begriff „Didaktik im weiteren Sinne“ sowohl die „Didaktik im engeren Sinn“ als auch die „Methodik“. Die Methodik ist somit eine Teildisziplin der Didaktik. Spricht man also heute von Didaktik, so meint man meist die Bildungsintention, die Auswahl der Bildungsinhalte, die Methoden und die Medien als Gesamtkomplex. (vgl. Riedl , 2004, S. 8f.)

II.4.2 Die Didaktik der Informatik

Durch die Zerteilung des Unterrichtes in die einzelnen Unterrichtsfächer sind unterschiedliche, pragmatisch orientierte, Fachdidaktiken entstanden. Die Didaktik der Informatik hat die Aufgabe konkrete Vorschläge für die Umsetzung der fachlichen Kenntnisse der Informatik zu entwickeln und deren Umsetzbarkeit zu evaluieren. (vgl. Humbert, 2006, S. 1)

II.4.3 Was ist guter Unterricht?

Mit dieser Frage beschäftigt sich der deutsche Pädagoge Hilbert Meyer. Er definiert zehn Merkmale des guten Unterrichtes wie folgt:

1. Klare Strukturierung des Unterrichts (Prozess-, Ziel-und Inhaltsklarheit; Rollenklarheit, Absprache von Regeln, Ritualen und Freiräumen)

2. Hoher Anteil echter Lernzeit (durch gutes Zeitmanagement, Pünktlichkeit; Auslagerung von Organisationskram; Rhythmisierung des Tagesablaufs)
3. Lernförderliches Klima (durch gegenseitigen Respekt, verlässlich eingehaltene Regeln, Verantwortungsübernahme, Gerechtigkeit und Fürsorge)
4. Inhaltliche Klarheit (durch Verständlichkeit der Aufgabenstellung, Plausibilität des thematischen Gangs, Klarheit und Verbindlichkeit der Ergebnissicherung)
5. Sinnstiftendes Kommunizieren (durch Planungsbeteiligung, Gesprächskultur, Sinnkonferenzen, Lerntagebücher und Schülerfeedback)
6. Methodenvielfalt (Reichtum an Inszenierungstechniken: Vielfalt der Handlungsmuster; Variabilität der Verlaufsformen und Ausbalancierung der methodischen Großformen)
7. Individuelles Fördern (durch Freiräume, Geduld und Zeit; durch innere Differenzierung und Integration; durch individuelle Lernstandsanalysen und abgestimmte Förderpläne; besondere Förderung von Schülern aus Risikogruppen)
8. Intelligentes Üben (durch Bewusstmachen von Lernstrategien, passgenaue Übungsaufträge, gezielte Hilfestellungen und „überfreundliche“ Rahmenbedingungen)
9. Transparente Leistungserwartungen (durch ein an den Richtlinien oder Bildungsstandards orientiertes, dem Leistungsvermögen der Schülerinnen und Schüler entsprechendes Lernangebot und zügige förderorientierte Rückmeldungen zum Lernfortschritt)
10. Vorbereitete Umgebung (durch gute Ordnung, funktionale Einrichtung und brauchbare Lernwerkzeuge)

(Meyer, 2011, S. 17f.)

Für die Abhaltung eines „guten“ Unterrichts sind alle zehn Merkmale von Meyer von Relevanz.

Für die Erstellung des Lehrerinnen- und Lehrerhandbuches sind für mich insbesondere die nachstehenden Punkte nach Meyer von Bedeutung.

Es ist besonderer Wert auf die „Inhaltliche Klarheit“, also auf die verständliche Beschreibung der Lerninhalte sowie der Aufgabenstellungen zu legen, da unklare Inhalte zu Missverständnissen und Missdeutungen führen können. Es muss darauf geachtet werden, dass die Wissensvermittlung aufbauend und nicht sprunghaft erfolgen kann. Wichtig ist, dass ein klarer „roter Faden“ für den Benutzer des Handbuches erkennbar ist. Die Ergebnissicherungen müssen laufend und umfassend stattfinden um zu gewährleisten, dass die Schülerinnen und Schüler schlussendlich selbstständig funktionierende Programme erstellen können.

Zu beachten sind weiters auch die Punkte „Individuelles Fördern“ und „Intelligentes Üben“. Es ist zu gewährleisten, dass unterschiedliche Schwierigkeitsgrade entsprechend der

jeweiligen Möglichkeiten der einzelnen Schülerinnen und Schüler angeboten werden. Durch Zusatzaufgaben können individuelle Anpassungen die Schülerinnen und Schüler gezielt gefordert und gefördert werden. Intelligentes Üben kann durch die Verwendung verschiedenster Unterrichtsmethoden umgesetzt werden. Bei der Erstellung der Übungsaufgaben ist dies entsprechend zu berücksichtigen wobei es den Lehrerinnen und Lehrern überlassen wird, welche Strategien situationsbezogen angewendet werden.

Zur weiteren Erkundung meiner Forschungsfrage widme ich mich im Folgenden dem sechsten Merkmal „Methodenvielfalt“ näher.

II.4.4 Unterrichtsmethoden und deren Vielfalt.

Den Unterrichtsmethoden/didaktischen Ansätzen liegen unterschiedliche Sozialformen zu Grunde. Diese sind im einzelnen Einzelarbeiten, Partnerarbeiten, Gruppenarbeiten oder Klassenunterricht.

Der Lernerfolg muss einerseits als quantitatives Maß des Lernzuwachses verstanden und gemessen und andererseits als qualitativ didaktisch bedeutsames Lernen begriffen werden. Wenn die didaktische Stimmigkeit des methodischen Vorgehens fehlt, dann wird das Lernen beliebig, wenn der empirisch erfassbare Effekt des methodischen Vorgehens fehlt, dann wird der Unterricht wirkungslos. Es sind also beide Aspekte des didaktischen Handelns von Bedeutung. Es gibt eine Vielzahl an Methoden und trotz des gegebenen Entwicklungsprozesses bei den Unterrichtsmethoden kann keine einen Anspruch darauf erheben, für alle didaktischen Gegebenheiten die Richtige zu sein. (vgl. Wiechmann & Wildhirt, 2016, S. 9)

Unter dem Begriff Unterrichtemethoden versteht man Planungs- und Umsetzungsmuster für die Gestaltung längerer und geschlossener Unterrichtssequenzen, wobei die kleinste Einheit eine Unterrichtsstunde darstellt. Eine derart geschlossene Sequenz setzt sich immer aus einer Aneinanderreihung mehrerer didaktischer Arbeitsschritte zusammen und schließt auch die entsprechende Sozialform ein. Man versteht darunter also nicht einzelne unterrichtsbezogene Handlungen wie zum Beispiel Vorträge, Übungen oder Demonstrationen. (vgl. Wiechmann & Wildhirt, 2016, S. 13)

Unterrichtsmethoden bieten die Möglichkeit zur erfahrungsbezogenen Weiterentwicklung des didaktischen Handelns der Lehrerinnen und Lehrer und sind unter drei Aspekten zu sehen:

1. Sie beschreiben den Weg zum Ziel des Unterrichtes und sind ein wesentlicher Bestandteil der Didaktik.
2. Sie ermöglichen eine Steigerung der Effektivität und der Effizienz des Unterrichtes.
3. Sie dienen der Planung, der kritischen Begleitung und der abschließenden Nachbearbeitung des Unterrichts.

(vgl. Wiechmann & Wildhirt, 12 Unterrichtsmethoden, 2016, S. 11f.)

Die Unterrichtsmethoden unterliegen einem ständigen Veränderungsprozess, welcher uns die heutige Vielfalt von Unterrichtsmethoden beschert. Dies begründet sich einerseits durch veränderte Aufgabenstellungen sowie Rahmenbedingungen des Unterrichtes sowie aus veränderten gesellschaftlichen Ansprüchen auf den Bildungsauftrag und andererseits durch deren Nutzung in der Folge eines pädagogischen Fortschrittes. (vgl. Wiechmann & Wildhirt, 2016, S. 12f.)

Es ist anzumerken, dass nicht jede Unterrichtsmethode für jede pädagogische Situation geeignet ist. Diese muss in Anbetracht auf die situativen Bedingungen, die Thematik, die Lerngruppe, die einzelnen Schülerinnen und Schüler, sowie auf die Erfahrungen und Kompetenzen der Lehrerinnen und Lehrer abgestimmt sein. Die Lehrrinnen und Lehrer können ihre didaktischen Intentionen nur umsetzen, wenn geeignete Unterrichtsmethoden verwendet werden. (vgl. Wiechmann & Wildhirt, 2016, S. 13)

Es sind drei Dimensionen bei der Auswahl der Unterrichtsmethoden zu beachten. (vgl. Wiechmann, 2002)

Die erste Dimension bezieht sich auf den Vermittlungsstil (normative Grundposition). Es steht dabei lehrendes Lernen im Gegensatz zum entdeckenden Lernen. Das entdeckende Lernen beschränkt sich dabei auf eine interessante Problemerörterung und die Bereitstellung von Lernmaterialien. Die Schülerinnen und Schüler werden dabei zu eigenen Entdeckungen angeregt und durch die Lehrerin oder den Lehrer begleitet. Beim lehrenden Lernen erfolgt die Wissensvermittlung an die Schülerinnen und Schüler durch Präsentationen der

Lehrerinnen und Lehrer. Angepasst an die Situation in der Klasse und die Vorlieben der Lehrerinnen und Lehrer können mehr oder minder entdeckende Elemente im Unterricht verwendet werden. Klare und eindeutige Inhalte, wie zum Beispiel die Grammatik einer Sprache, sind meist durch die Verwendung des lehrenden Unterrichtes effektiver zu vermitteln. Für offene Themen, wie zu Beispiel die Religionen, ist auch der entdeckende Unterricht geeignet. (vgl. Wiechmann & Wildhirt, 2016, S. 19)

Die zweite Dimension bezieht sich auf die Unterrichtssteuerung. Es steht dabei der lehrergelenkte Unterricht dem schülergelenkten Unterricht gegenüber, woraus sich aber eine Diskrepanz auftut. Dies ist schon darin erkennbar, dass die durchzuführende Unterrichtsvorbereitung und Unterrichtsplanung durch die Lehrkraft erfolgt. (vgl. Wiechmann & Wildhirt, 2016, S. 19) Ein Dilemma besteht auch darin, dass die Lehrerin oder der Lehrer die Schülerinnen und Schüler auch mit Gewalt zur Selbstständigkeit führen muss. Auf der Seite der Schülerinnen und Schüler besteht der Widerspruch darin einerseits selbstständig handeln zu sollen und andererseits auf die Hilfe der Lehrerin oder des Lehrers angewiesen zu sein. (vgl. Meyer, 2009, S. 55) Zusätzlich sind die Schülerinnen und Schüler auf die Bewertungen der Lehrerinnen und Lehrer angewiesen, da keine Selbsteinschätzung der Leistungen der Schülerinnen und Schüler erfolgt. (vgl. Rogers, 1983) Elemente der Grundbildung wie zum Beispiel Termumformungen in der Mathematik eignen sich für eine fremdbestimmte (schülergelenkte) Unterrichtsform. Diese Wissensvermittlung gelingt effektiver, wenn sich die Schülerinnen und Schüler aus eigener Initiative mit diesem Thema auseinandersetzen. (vgl. Wiechmann & Wildhirt, 2016, S. 19)

Die dritte Dimension bezieht sich auf die Unterrichtsgestaltung und spannt sich als Diagonale eines Entscheidungsfeldes zwischen den Dimensionen Vermittlungsstil und Unterrichtssteuerung auf. Es steht der planvolle Unterricht dem situativen Unterricht gegenüber. Während Johann Friedrich Herbart Anfang des 19. Jahrhunderts als Klassiker der Pädagogik die Meinung vertrat, dass erfahrungsorientiertes (situatives) Lernen aus der Schule fernzuhalten ist, sind nachfolgende Pädagogen wie Friedrich Copei (1969) der Ansicht, dass fruchtbringende Bildungsprozesse nur durch flexibles und situatives Reagieren möglich werden. (vgl. Wiechmann & Wildhirt, 2016, S. 19)

II.4.5 Unterrichtsmethoden

Im Rahmen der Erstellung dieser Diplomarbeit bin ich in umfangreichen Recherchen auf eine Vielzahl an didaktischen Ansätzen beziehungsweise Unterrichtsmethoden gestoßen. In der Folge beschäftigt sich diese Diplomarbeit mit einzelnen Unterrichtsmethoden beziehungsweise didaktischen Ansätzen, welche mir für die Wissensvermittlung zum Thema Einführung in die VBA Programmierung mit Microsoft Excel geeignet erscheinen.

II.4.5.1 Frontalunterricht

Der Frontalunterricht definiert sich nach Schaub und Zenke wie folgt:

> Sozialform des Unterrichts, bei dem ein Lehrer versucht, den Lernstoff an eine Schulklasse mit Hilfe sprachlicher Darbietung, Wandtafel, Schulbuch und Overheadprojektor unter Berücksichtigung methodischer Lernschritte an alle Schüler gleichzeitig und effektiv zu vermitteln. Dabei steuert und kontrolliert er mit Fragen und Impulsen den Fortgang des Lernprozesses. (Schaub & Zenke, 2000, S. 224)

Der Frontalunterricht gehört neben der Schüler-Interaktion zur Sozialform des Klassenunterrichtes. Der Frontalunterricht ist eine frontale und lehrergelenkte Interaktion zwischen Schülerinnen und Schülern mit Lehrerinnen und Lehrern, wobei jeweils die ganze Klasse angesprochen wird. (vgl. Gudjons, 2011, S. 23)

Laut Gudjons existieren zwei Formen des Frontalunterrichtes. Er unterscheidet einerseits dabei den traditionellen Frontalunterricht, bei welchem eine methodische Monokultur im Vordergrund steht und andererseits als modernere Form den integrierten Frontalunterricht. Dabei werden die frontalunterrichtlichen Phasen durch selbstgesteuerte Schülerarbeitsformen ergänzt, sodass die Schüler Eigentätigkeit, Selbststeuerung sowie Selbstverantwortung übernehmen. (vgl. Gudjons, 2011, S. 24)

In der Folge gehe ich in meiner Diplomarbeit auf das Zweitgenannte, den integrierten Frontalunterricht, ein.

Dieser Frontalunterricht besteht aus vier Phasen, welche zwingend aufeinander aufbauen. Dies sind „Darbieten“, „Konstruktives Durcharbeiten“, „Übendes Wiederholen“ und „Problemorientiertes Anwenden“. (vgl. Wiechmann & Wildhirt, 2016, S. 28)

Beim „Darbieten“ erfolgt die Vermittlung des Grundlagenwissens. Dies kann einerseits durch ein informierendes Darbieten in der Form eines Referates der Lehrerin beziehungsweise des Lehrers oder durch ein problemorientiertes Darbieten erfolgen, wobei ein Sachthema als Problem dargestellt wird. Festzuhalten ist, dass die problemorientierte Darbietung zwar die schwierigere, aber auch die effektivere Variante ist. Die Problemstellung muss dabei für die Schülerinnen und Schüler tatsächlich relevant sein, damit ein sachbezogenes Interesse entsteht. (vgl. Wiechmann & Wildhirt, 2016, S. 28ff.)

Beim Punkt „Konstruktives Durcharbeiten“ muss das vorher vermittelte Grundlagenwissen in die bestehenden kognitiven Strukturen der Schülerinnen und Schüler eingegliedert werden. Wichtig dabei ist, dass sich die Schülerinnen und Schüler selbst mit dem Thema auseinandersetzen müssen, was zum Beispiel mit Frage-Antwortspielen gut gelingt. Es kann dabei das Thema aus unterschiedlichen Perspektiven beleuchtet werden, wobei der Schwierigkeitsgrad dabei genau angepasst sein muss. Die Schülerinnen und Schüler müssen dabei mit einer individuellen Auseinandersetzung zu einer Verbalisierung des Themas aufgefordert werden. Zu beachten ist, dass es in dieser Phase keine falschen Antworten gibt, höchstens schlechte Fragen. Durch die jeweiligen Antworten erhalten die Lehrenden eine sofortige und präzise Rückmeldung, ob die Schülerinnen und Schüler das zu erarbeitende Thema auch verstanden haben. Aber auch die Schülerinnen und Schüler profitieren von der Rückmeldung der Lehrerinnen und Lehrer, wodurch der Lerneffekt sehr hoch ist. Gleichzeitig werden alle Schülerinnen und Schüler in den Unterricht integriert, entweder selbstmeldend oder durch Aufforderung der Lehrenden. Wichtig dabei ist zu beachten, dass die Schülerinnen und Schüler diese Phase unterschiedlich schnell bewältigen. Schnelle Schülerinnen und Schüler können früher in die dritte Phase übergehen, wodurch für die Lehrenden der Freiraum entsteht, sich gezielter mit den langsameren Schülerinnen und Schüler zu beschäftigen. (vgl. Wiechmann & Wildhirt, 2016, S. 31f.)

In der dritten Phase „Übenden Wiederholens“ wird das flüchtige Verständnis des Themas durch individuelle Übungen gefestigt. Die Schülerinnen und Schüler sind dabei gefordert ihr erlangtes Wissen neu zu strukturieren was bedingt, dass das durch die Schülerinnen und Schüler einzeln erfolgt. Die größte Schwierigkeit liegt dabei darin, den Schülerinnen und Schülern die richten Übungsaufgaben zu geben. Zu diesem Zwecke sollten differenzierte Übungsaufgaben (unterschiedliche Schwierigkeitsgrade) zur Verfügung gestellt werden. Durch eine freie Wahl des Schwierigkeitsgrades erhalten die Schülerinnen und Schüler ein

gewisses Maß an Autonomie, was zur Stärkung der Motivation führt. Es muss dabei darauf geachtet werden, dass keine Fehler eingelernt werden. Die Schülerinnen und Schüler müssen Rückmeldungen über die Korrektheit der Übungen erhalten. Gegebenenfalls können auch Übungen mit einer möglichen Selbstkontrolle zum Ziel führen. Zusätzlich ist es erforderlich den Schülerinnen und Schülern bestmögliche Unterstützung zukommen zu lassen, entweder durch Helfersysteme oder durch die Lehrenden selbst. Die Schülerinnen und Schüler müssen durch die Übungen das Gefühl eines Kompetenzzuwachses haben. Es ist dabei zu beachten, dass die Lernenden mindestens 80 % der Aufgaben richtig lösen können. Ein Überlernen sichert die Nachhaltigkeit des angeeigneten Wissens, kann aber bei einer Überstrapazierung zu einer Demotivierung der Schülerinnen und Schüler führen. (vgl. Wiechmann & Wildhirt, 2016, S. 32ff.)

Die vierte Phase, das „Problemorientierte Anwenden“, trägt dazu bei, die vorher erworbenen kognitiven Kompetenzen aus dem Kontext herauszulösen und auf andere Probleme zu übertragen. Dies erfordert, dass die Aufgaben gegenüber der Übungsphase grundlegend anders sein müssen. Es müssen die Aufgaben als echte Probleme verstanden werden, Lösungsansätze dürfen nicht gleich erkennbar sein. Der Schwierigkeitsgrad kann durch den Grad der Ähnlichkeit der gestellten Aufgaben zu den vorangegangenen Übungen verändert werden. Als Sozialform bieten sich für diese Phase Partner- oder Gruppenarbeiten an. Eine erfolgreiche Problemlösung stärkt bei Schülerinnen und Schülern die Motivation um sich weiteren Problemen in neuen Kontexten zu stellen. Bei Hilfestellungen dürfen aber nur Lösungswege aufgezeigt werden, jedoch keine Lösungen. Am Ende der Phase ist es unbedingt erforderlich, dass die Ergebnisse der Schülerinnen und Schüler durch die Lehrenden anerkannt und gewürdigt werden. (vgl. Wiechmann & Wildhirt, 2016, S. 34ff.)

Die Vorteile dieser Unterrichtsform liegen darin, dass diese sehr zeitökonomisch, gut planbar und effektiv ist. Die Lehrkraft kann den Unterricht optimal lenken, hat alle Schülerinnen und Schüler im Blick und kommunikative Fähigkeiten werden geübt. Es bedarf keiner langen Erklärungen, da die Schülerinnen und Schüler diese Unterrichtsform gut kennen.

Nachtteile sind insoweit erkennbar, dass der Unterricht sehr Lehrenden konzentriert abläuft und der Lernerfolg deshalb sehr von deren oder dessen Fähigkeiten abhängt. Weiters werden die Schülerinnen und Schüler nur sehr selten aktiv, es fehlt eine interaktive Komponente und

ein sozialerzieherischer Effekt, wie er zum Beispiel bei Gruppenarbeiten gegeben ist. (vgl. Schlimok, Frontalunterricht – besser als sein Ruf, 2015)

II.4.5.2 Direkte Instruktion

Die Unterrichtsmethode „Direkte Instruktion“ wird im deutschen Sprachgebrauch oftmals auch „Direktes Unterrichten“ oder „Direkte Unterweisung“ genannt.

Bei der direkten Instruktion haben die Lehrerinnen und Lehrer eine zentrale Rolle inne. Das Unterrichtsgeschehen wird zur Gänze von ihnen bestimmt. Diese Methode widerspricht heutigen Werten wie zum Beispiel der Selbstständigkeit oder Eigenverantwortung der Schülerinnen und Schüler, weshalb die allgemeine Akzeptanz sehr gering ist. Diese Unterrichtsmethode ist sicherlich nicht für alle didaktischen Aufgaben die richtige Wahl. Zur Vermittlung von Grundlagenwissen ist sie aber sehr brauchbar und effektiv. (vgl. Wiechmann & Wildhirt, 2016, S. 40)

Wichtig ist, dass die Lehrerin oder der Lehrer den didaktischen Prozess kompakt und geradlinig umsetzen damit die Schülerinnen und Schüler aus dem Lernprozess nicht aussteigen können. (vgl. Wiechmann & Wildhirt, 2016, S. 41)

Die Unterrichtsmethode „Direkte Instruktion“ gliedert sich in die drei Hauptphasen „Präsentation/Demonstration“, „Gemeinsames Üben“ und „Individuelles Üben“ wobei eine „Aktualisierungsphase“ vorangeht und am Ende der Stunde eine sogenannte „Bilanz“ steht. (vgl. Wiechmann & Wildhirt, 2016, S. 42ff.)

Zu Beginn wird den Schülerinnen und Schülern in der Aktualisierungsphase mitgeteilt, was sie im Laufe der Unterrichtseinheit lernen werden. Danach wird in der Phase der Präsentation das neue Thema demonstriert beziehungsweise präsentiert, wobei dies immer in kleinen und aufeinander aufbauenden Schritten erfolgt, bis schlussendlich das gesamte Thema dargestellt ist. Allfällige Abschweifungen sollten dabei vermieden werden. Erklärungen müssen ausführlich sein. Um das richtige Tempo beizubehalten muss die Lehrkraft regelmäßig ein Feedback hinsichtlich des Lernfortschrittes einholen. Die Präsentation ist nur erfolgreich, solange die Aufmerksamkeit der Schülerinnen und Schüler aufrechterhalten werden kann. Sinkt diese, so sollte diese beendet werden und zu den nächsten Phasen übergegangen werden. (vgl. Wiechmann & Wildhirt, 2016, S. 43)

In der Phase des „gemeinsamen Übens“ wird aktiv und gemeinsam geübt, um das vorher Kennengelernte zu lernen. Die Übung läuft so ab, dass der Lehrer oder die Lehrerin viele und kurze Fragen stellt, welche eine Schülerin oder Schüler beantwortet. Die Lehrerin oder der Lehrer gibt unmittelbar ein präzises Feedback bezüglich der Richtigkeit und korrigiert gegebenenfalls in kurzer Form unter Bezugnahme auf die Präsentation. Dies wiederholt sich so lange, bis alle die richtigen Antworten geben. Nach den anfänglichen Einzelantworten ist ein Wechsel auf Gruppenantworten möglich. In die Phase des individuellen Übens darf erst übergegangen werden, wenn das neu erworbene Wissen gesichert ist, da ansonsten falsches Wissen verfestigt werden würde. (vgl. Wiechmann & Wildhirt, 2016, S. 43ff.)

In der Phase des „individuellen Übens“ erfolgt die Sicherung und Automatisierung des neu erworbenen Wissens. Die Schülerinnen und Schüler müssen deren erworbenes Wissen ohne unmittelbares Feedback der Lehrerin oder des Lehrers verwenden. Problematisch ist in dieser Phase, dass mangels eines sofortigen Feedbacks nicht korrigierte Fehler fleißig geübt werden. (vgl. Wiechmann & Wildhirt, 2016, S. 45f.)

Zum Abschluss erfolgt eine kurze Bilanz über die Unterrichtseinheit. Durch diese bewusste Wahrnehmung stellt sich bei den Schülerinnen und Schülern ein Gefühl der Zufriedenheit ein, sie registrieren, dass sich die Anstrengungen gelohnt haben, was wiederum motivierend wirkt. (vgl. Wiechmann & Wildhirt, 2016, S. 46)

II.4.5.3 Pair-Programming

Unter „Pair-Programmierung“ versteht man einen Programmierstil, bei welchem zwei Programmierer gleichzeitig und nebeneinander an einem Rechner am selben Programm arbeiten. Dabei sind beide Personen gleichberechtigt, einer ist der „Pilot“ und der andere der „Navigator“. Der Pilot hat die Aufgabe den Programmcode zu erstellen, der Navigator kontrolliert und überprüft. In regelmäßigen Abständen kann die Position geändert werden. Bei größeren Programmierarbeiten empfiehlt es sich auch wenn die Paare verändert werden. (vgl. Hölscher, 2018)

Was auf den ersten Blick als Ressourcenvergeudung erscheint, ist bei näherer Betrachtung jedoch sehr effektiv und ökonomisch. Es ist erwiesen, dass jeder Programmcode Fehler enthält, welche im Nachhinein sehr aufwändig gesucht werden müssen. (vgl. Hölscher, 2018)

Die Pair-Programmierung bringt einige Vorteile mit sich. Die Code-Qualität steigt, weil Fehler oftmals direkt erkannt werden und eine Nachbearbeitung so vermieden werden kann. Weiters wissen zwei Programmierer über den Programmcode und die Arbeitsweise des Programmes genau Bescheid, in der Wirtschaft ist dies sehr von Vorteil, zum Beispiel bei Urlaubsvertretungen. Es wird die Know-How-Verteilung verbessert, da das Wissen nicht nur in einer Person gebündelt ist. Die Arbeit wird auch effektiver, da sich das Paar selbst anspornt und vorantreibt, während nebenbei die sozialen Kontakte gefördert werden. Den Einwand, dass die zwei Mitglieder eines Paares miteinander nicht „Können“, kann man nicht gelten lassen. Derartige Probleme müssen im Vorfeld auf einer anderer Ebene gelöst werden. (vgl. Hölscher, 2018)

Bezogen auf den Unterricht in der Schule bedeutet dies, dass der große Wert im „voneinander Lernen“ zu suchen ist. Um zu vermeiden, dass die Programmierarbeit nur von einer Person geleistet wird, muss eine Einweisung der Schülerinnen und Schüler durch die Lehrenden erfolgen. Weiters empfiehlt es sich die Paare durch den Lehrenden zusammen zu stellen. Es wird so vermieden, dass zwei ganz gute oder zwei ganz schlechte Schülerinnen und Schüler sich in einem Paar finden. (vgl. Khan Academy, 2018)

II.4.5.4 Einzelarbeit

Wesentliche Merkmale der Einzelarbeit sind, dass die Schülerinnen und Schüler still, eigenverantwortlich uns selbstständig arbeiten. Sie bestimmen das Tempo sowie die Reihenfolge ihrer Arbeitsschritte eigenständig und sind frei in ihren Denkprozessen. (vgl. Drumm, 2007, S. 18)

Durch den Einsatz von Einzelarbeiten werden individuelle Denkmuster ermöglicht, was bei der Aufnahme und Einübung neuen Lernstoffes sowie neuer Techniken förderlich ist. Durch die Einzelarbeiten wird bei den Schülerinnen und Schülern die Selbstständigkeit, die Konzentrationsfähigkeit und das Durchhaltevermögen gestärkt. (vgl. Drumm, 2007, S. 18)

Einzelarbeiten eignen sich neben dem Schreiben von Tests und Hausaufgaben insbesondere auch für die Erarbeitung neuer Inhalte, zur kreativen Vertiefung bereits erworbener Kenntnisse und zur Übung bereits erarbeiteten Stoffes. (vgl. Drumm, 2007, S. 18)

Vom zeitlichen Aspekt her können Einzelarbeiten von wenigen Minuten bis zu mehreren Stunden in Anspruch nehmen. Die Durchführung von Einzelarbeiten unterteilt sich in drei Phasen, die Einführungsphase, die Arbeitsphase und die Kontrollphase.
In der Einführungsphase müssen genaue Anweisungen über die abzuarbeitenden Aufgaben, das Verhalten während der Arbeitsphase und die zur Verfügung stehende Zeit gemacht werden. Während der Arbeitsphase sollte die Lehrerin oder der Lehrer zwar prinzipiell als Ansprechperson zur Verfügung stehen, konkrete Problemlösungen sollten sie jedoch nicht anbieten, sondern nur Lösungswege aufzeigen. In der dritten Phase, der Kontrollphase, werden Lösungen präsentiert. Dies kann auch durch die Schülerinnen und Schüler selbst erfolgen. Durch die Lehrerinnen oder Lehrer kann eine Musterlösung präsentiert werden. (vgl. Drumm, 2007, S. 20ff.)

Zusammenfassend können die Vorteile von Einzelarbeiten wie folgt beschrieben werden. Einzelarbeiten sind eine sehr effektive Arbeitsform, bei welcher die Schülerinnen und Schüler, gegebenenfalls durch differenzierte Aufgabenstellungen auf unterschiedlichen Niveaus, in ihrem eigenen Tempo arbeiten können. Die Lehrerinnen und Lehrer erhalten durch die Arbeitsergebnisse eine Rückmeldung über den Leistungsstand der einzelnen Schülerinnen und Schüler. Als Nachteil sei erwähnt, dass die Teamfähigkeit durch diese

Unterrichtsmethode nicht gefördert wird. (vgl. Schlimok, Einzelarbeit oder Stillarbeit – Voraussetzung für andere Sozialformen im Unterricht, 2015)

II.4.6 Mögliche Umsetzung des Lehrerinnen- und Lehrerhandbuches

Von den vorher genannten Unterrichtsmethoden können mehrere Ansätze bei der Umsetzung der Wissensvermittlung, auf Basis des in der Anlage befindlichen Lehrerinnen- und Lehrerhandbuches, zur Anwendung gelangen.

In der Anfangsphase, zur Vermittlung des Grundlagenwissens, empfiehlt sich die Unterrichtsmethode des „Frontalunterrichtes“. In der ersten Phase der Darbietung kann meiner Meinung nach das Setzen der allgemeinen Einstellungen im System, die Vorstellung der Benutzeroberfläche und die Vermittlung des Basiswissens über Methoden, Eigenschaften, Prozeduren und Funktionen, die Objekthierarchien, sowie die Sprachelemente in Visual Basic for Applications (VBA) am effektivsten vermittelt werden. Die Strukturen der Sprachelemente sind im Großen und Ganzen, bis auf optionale Ergänzungen, immer gleich. Es stehen Übungen zum gemeinsamen und zum alleine Üben zur Verfügung. In der Phase des konstruktiven Durcharbeitens und des übenden Wiederholens können diese abgearbeitet werden. Der Schwierigkeitsgrad kann durch das Hinzufügen oder Weglassen von Detailaufgaben leicht angepasst werden. In der vierten Phase des problemorientierten Anwendens stehen drei Musteraufgaben zur Verfügung. Ein eventueller Ansatz wäre dabei ein Programm in Lehrer-Schüler-Interaktion, eines in „Pair-Programming“ und das letzte als „Einzelarbeit“ durchzuführen. Der Vorteil liegt dabei darin, dass die Schülerinnen und Schüler nicht sofort auf sich alleine gestellt sind, sondern gemeinschaftlich erste Lösungen gesucht werden. Es besteht bei zu vielen Misserfolgen die Gefahr, dass dadurch die Schülerinnen und Schüler demotiviert werden.

Alternativ ist für diese erste Wissensvermittlung des Basiswissens auch die Unterrichtsmethode „Direkte Instruktion“, zumindest Phasenweise, wie zum Beispiel bei der Wissensvermittlung hinsichtlich der Sprachelemente in Visual Basic for Applications (VBA), möglich. Beim gemeinschaftlichen Üben können Befehle durch die Lehrerin oder dem Lehrer genannt werden, welche jeweils eine Schülerin oder ein Schüler erklären muss. Es ist auch denkbar, dass ein Programmcode gezeigt wird und die Schülerinnen und Schüler

der Reihe nach jeweils eine Programmzeile kurz erläutern müssen. Es ist auch denkbar, dass statt Einzelantworten in dieser Phase Gruppenantworten akzeptiert werden. Die Phase des individuellen Übens würde ich analog zur problemorientierten Anwendung des Frontalunterrichtes durchführen.

In beiden Fällen erfolgt eine effektive Vermittlung des Basiswissens. Die Schülerinnen und Schüler können mittels leicht differenzierbaren Übungsaufgaben die aus den Darbietungen oder Präsentationen erworbenen Kenntnisse festigen, verinnerlichen und aus dem Kontext isolieren, um diese bei späteren Problemlösungen anwenden zu können.

Die Umsetzung des „Problemorientierten Anwendens“ bei der Unterrichtsmethode „Frontalunterricht“ beziehungsweise des „individuellen Übens“ bei der Unterrichtsmethode „Direkte Instruktion“ könnte zuerst in Gruppenarbeiten und anschließend in Einzelaufgaben umgesetzt werden.

Für die ersten Gruppenarbeiten bietet sich das „Pair-Programming“ an. Die zwei Schülerinnen oder Schüler eines Paares kontrollieren sich quasi gegenseitig und machen sich auf Fehler aufmerksam. In dieser Phase der ersten Umsetzungen ist es sehr förderlich, wenn sich zwei Schülerinnen oder Schüler bei aufkommenden Problemen gegenseitig motivieren und diese gegebenenfalls gemeinsam an Lösungsansätzen arbeiten. Die Gefahr des Resignierens und Aufgebens bei Misserfolgen, welche mit nahezu hundertprozentiger Wahrscheinlichkeit in der Anfangsphase des Programmierens auftreten werden, sinkt dadurch bei den Schülerinnen und Schülern. Gleichzeitig werden auch soziale Kontakte in der Klasse gefördert.

Durch die folgende Einzelarbeit müssen die Schülerinnen und Schüler anschließend beweisen, dass diese auch selbstständig in der Lage sind gegebene Aufgaben in Programme umzusetzen. In diesem Zusammenhang werden die Schülerinnen und Schüler immer wieder auf neue Probleme stoßen, welche einer Lösung bedürfen. In diesem Zusammenhang ist zu erwähnen, dass sich Einzelarbeiten neben der Sicherung und Vertiefung bestehenden Wissens auch für die Erarbeitung neuer Inhalte eignen. Durch die fast unendliche Anzahl an unterschiedlichen Lösungsansätzen und Umsetzungsmöglichkeiten sollte jede Schülerin und Schüler einen Ansatz finden können. In dieser Phase der Einführung in die Programmierung sollten die Schülerinnen und Schüler nicht damit belastet werden Programme mit optimaler Performance zu entwickeln. Vielmehr sollten die erlernten Fähigkeit die Aufgabe zu

verstehen, zu analysieren und in verwertbare Algorithmen umzusetzen in den Vordergrund gestellt sein.

Bei der vorstehend skizzierten möglichen Umsetzung des Lehrerinnen- und Lehrerhandbuches ist in der Anfangsphase durch den „Frontalunterricht“ beziehungsweise die Unterrichtsform der „direkten Instruktion“ die effektive Vermittlung von Basiswissen gewährleistet. Durch das folgende „Pair-Programming“ erfolgt eine Wissensvertiefung in einer sozialen Umgebung, wodurch das Risiko der Demotivierung durch Misserfolge vermindert wird. Durch die abschließende Einzelarbeit erhalten die Lehrerinnen und Lehrer ein Feedback über den Leistungsstand der einzelnen Schülerinnen und Schüler, welches auch in die anschließende Beurteilung einfließen kann.

III. Praktischer Teil

III.1 Entstehung und Aufbau des Lehrerinnen- und Lehrerhandbuches

In diesem Teil meiner Diplomarbeit habe ich ein Handbuch für Lehrerinnen und Lehrer für den Einstieg in die Programmierung mit Visual Basic for Applications (VBA) mit Microsoft Excel erstellt.

Dieses Handbuch entstand nach einer eingehenden und ausführlichen Literaturrecherche.

Eingangs wird in dem Handbuch erklärt, was Visual Basic for Applications (VBA) ist und welche vorbereitende Arbeiten und Einstellungen im Microsoft Excel zu tätigen sind. Es werden Begriffe und der Umgang mit Makros erklärt. Im Folgenden beschäftigt sich das Handbuch mit der Visual Basic Entwicklungsumgebung (VBE) und mit deren Bestandteilen, der Menüleiste, der Symbolleiste, dem Projekt-Explorer, dem Code-Fenster, dem Direktfenster und dem Objektkatalog.

Es sind erste Interaktionen zum Kennenlernen der Benutzeroberfläche ebenso enthalten wie Beschreibungen von Methoden, Eigenschaften und Ereignissen, sowie der Hierarchie der Objekte in Visual Basic for Applications (VBA).

Nach einer anschließenden Beschreibung hinsichtlich der Verwendung von Variablen und Arrays sowie von Prozeduren und Funktionen werden die wichtigsten Sprachelemente von Visual Basic for Applications (VBA) dargestellt.

Abgeschlossen wir das Handbuch für Lehrerinnen und Lehrer mit drei Musteraufgaben, bei welchen ein besonderer Wert auf einen Alltagsbezug sowie auf eine allfällige Verwendung im fächerübergreifenden Unterricht gelegt wurde. Bei den Musterprogrammen stehen jeweils erklärende Hinweise für die Lehrenden, eine konkrete Aufgabenstellung, eine Musterlösung und Zusatzerläuterungen zu den Musterprogrammen zur Verfügung.

Das Lehrerinnen- und Lehrerhandbuch finden Sie Anhang dieser Diplomarbeit.

III.2 „Proof of Concept“

III.2.1 Das gewählte Vorgehen

Die Überprüfung der Verwendbarkeit des Lehrerinnen- und Lehrerhandbuches im Unterricht erfolgt mittels Begutachtung durch Lehrerinnen und Lehrer.

Aus dem Feedback soll erkennbar werden, ob der Aufbau und die Struktur des Handbuches dazu geeignet sind, um dieses im Unterricht einzusetzen. Es sollen dabei die Verständlichkeit und die Klarheit der Darbietung der Theorie beurteilt werden. Hinsichtlich der Beispielaufgaben ist von Interesse, ob diese in ausreichender Anzahl, mit dem entsprechenden Schwierigkeitsgrad und mit einem Realitätsbezug erstellt wurden. Darüberhinausgehend befinden sich noch drei Kernfragen über Visual Basic for Applictions, Angaben zur Person der überprüfenden Lehrerinnen und Lehrer sowie eine offene Frage bezüglich weiterer Tipps und Hinweise in dem Fragebogen.

Die Fragen sind im Bereich von „stimme nicht zu“ bis „stimme zu“ in vier Stufen zu beantworten. Es wird absichtlich darauf verzichtet einen Mittelwert anzubieten, um eine eindeutig „positive“ („stimme eher zu“ beziehungsweise. „stimme zu“) oder eine „negative“ („stimme nicht zu“ beziehungsweise. „stimme eher nicht zu“) Aussage zu erhalten. Es besteht aber auch die Möglichkeit „keine Antwort“ zu geben. Für diesen Fall oder für weitergehende Verbesserungsvorschläge werden Freitextfelder zur Verfügung gestellt.

Zur Umsetzung des „Proof of Concepts“ wird der Entwurf des Handbuches, unter Beifügung eines Fragebogens (siehe dazu den Anhang zu dieser Diplomarbeit), mit der Bitte um konstruktives Feedback übergeben, wobei mein Vorgehen dabei wie folgt zu beschreiben ist.

Es wurden durch mich zwei Lehrerinnen und drei Lehrer mit einer einschlägigen Informatikausbildung und einer Berufserfahrung von mindestens zehn Jahren persönlich kontaktiert. Weitere zwei Lehrerinnen wurden über die durch mich kontaktierten Lehrerinnen und Lehrer ebenfalls um deren Begutachtung und Feedback ersucht. Die Lehrpersonen unterrichten an zwei unterschiedlichen allgemeinbildenden höheren Schulen (AHS) in den Bezirken Mattersburg und Eisenstadt sowie in der Höheren technischen Bundeslehranstalt (HTBLA) in Eisenstadt. In Summe wurden somit sieben Lehrerinnen und

Lehrer, welche an drei unterschiedlichen höheren Schulen das Unterrichtsfach Informatik unterrichten, um deren Feedback ersucht.

III.2.2 Das Feedback, Daten und Ergebnisse

Insgesamt haben mir vier (jeweils zwei Lehrerinnen und Lehrer) von den sieben damit konfrontierten Lehrenden, welche neben dem Unterrichtsfach Informatik noch die Fächer Mathematik, Physik, Biologie unterrichten, ein auswertbares Feedback gegeben, welches jeweils in der Folge durch mich analysiert und durch eine entsprechende Adaptierung des Handbuchs verwertet wurde. Diese vier Lehrenden geben an bereits „Coding“ unterrichtet zu haben, wobei drei davon auch schon Visual Basic for Applications (VBA) gelehrt haben. Diese drei Lehrerinnen und Lehrer schätzen deren eigene Kenntnisse in der VBA-Programmierung nach dem Schulnotenprinzip zwei Mal mit „Sehr gut“ und einmal mit „Sehr gut“ bis „Gut“ ein.

Der Fragenblock bezüglich des Aufbaues und der Struktur des Handbuches, somit die Übersichtlichkeit, die Strukturierung, die Reihenfolge der Themen sowie die Anzahl der enthaltenden Abbilddungen und deren Aussagekraft, wurde durchgehend mit der Bewertung „stimme zu“ und somit „positiv“ bewertet. Verbale Verbesserungsvorschläge wurden zu diesem Punkt keine angeführt.

Der Teil bezüglich der Verständlichkeit und der Klarheit des Handbuches wurde in Summe ebenfalls als „positiv“ bewertet. Die Verständlichkeit der Formulierungen der Erklärungen und die Nachvollziehbarkeit der Schritte wurde durchwegs mit „stimme zu“ eingestuft, wogegen die Länge der Erklärungen und die Vollständigkeit der Theorie mit „stimme zu“ bzw. „stimmt eher zu“ etwas schlechter bewertet wurden. Als Verbesserungsvorschläge wurde angeregt die Länge der Erklärungen etwas zu reduzieren und zu straffen. Zusätzlich wurde noch das Fehlen einer Gesamtübersicht der zur Verfügung stehenden VBA Befehle samt einer Kurzbeschreibung beanstandet und die Anführung hilfreicher Internetseiten mit weiterführenden Beschreibungen angeregt. Es wurde weiters bemängelt, dass die Theorie bezüglich der in VBA zur Verfügung stehenden Operatoren sowie deren Arbeitsweise und Wertigkeit der Abarbeitung im Handbuch für Lehrerinnen und Lehrer nicht enthalten sind.

Die Anzahl der im Handbuch enthaltenen Beispiele wurde durchgehend mit „stimme zu“ und somit „positiv“ eingestuft. Die Schwierigkeitsgrade der Beispiele, die dazugehörenden

Formulierungen bei den Erklärungen, deren fächerübergreifende Verwendbarkeit und Realitätsnähe wurden zwar in Summe positiv, mit den Kategorien „stimme eher zu“ und „stimme zu“ jedoch etwas kontroverser bewertet. Bei der Frage nach dem Schwierigkeitsgrad wurde einmal „keine Antwort“ gegeben und somit keine Einstufung durchgeführt. Es wurde angemerkt, dass die Angemessenheit des Schwierigkeitsgrades wohl in Verbindung mit der Schulstufe der zu unterrichtenden Schülerinnen und Schüler zu sehen ist. Es wurde angeregt Beispiele zu wählen, welche weniger Mathematik lastig sind und somit weniger Bezug zur Mathematik haben um die Schülerinnen und Schüler in der Phase des Erlernens des Programmierens nicht noch zusätzlich damit zu belasten. Bezüglich der Erklärungen der Beispiele wurde vorgeschlagen die Beispiele mit zusätzlichem „Input“ und somit mit weiteren Zusatzinformationen zu versehen.

Der Frageblock mit den Kernfragen, einerseits zur Einschätzung der Eignung von VBA um das „Coding“ zu erlernen und andererseits bezüglich der Bereitschaft das Handbuch im Unterricht selbst zu verwenden beziehungsweise dieses Handbuch anderen Lehrerinnen und Lehrern weiter zu empfehlen, wurde in den Bereichen „stimmt eher zu“ und „stimmt zu“ in Summe positiv bewertet.

In der abschließenden offenen Frage nach weiteren Tipps und Hinweisen wurde angeregt auch Materialen für die Schülerinnen und Schüler zu entwerfen. Dies könnten zum Beispiel Handouts über die wesentlichen Teile der Theorie von VBA oder auch Aufgabenblätter sein.

III.2.3 Einarbeitung des Feedbacks aus dem „Proof of Concept“ in das Lehrerinnen- und Lehrerhandbuch

Auf Grund der positiven Rückmeldung bezüglich des Aufbaues und der Struktur des Handbuches wurde dieses in der Endfassung diesbezüglich nur geringfügig angepasst. Es wurde eine Aufstellung der gängigsten Befehle eingefügt, wobei die angeführten Befehle jeweils mit einem Kurzkommentar versehen sind

Zur Vervollständigung der Theorie wurde eine detaillierte Beschreibung der in Visual Basic for Applications zur Verfügung stehenden Operatoren in das Handbuch aufgenommen. Weiters wurden Musterprogramme zur Veranschaulichung der Arbeitsweise der Operatoren erstellt und beschrieben.

Das komplette Handbuch wurde in Bezug auf eine mögliche Verkürzung der Erklärungen und Beschreibungen durchforstet. Es konnten zahlreiche Komprimierungen und Kürzungen vorgenommen werden, welche den Lesefluss verbessern und die Übersichtlichkeit erhöhen.

Die Beispielaufgaben wurden hinsichtlich der Formulierung der Aufgabenstellungen überarbeitet. Weiters wurden jeweils Zusatzerläuterungen zu den Musterprogrammen hinzugefügt, um die Nachvollziehbarkeit zu verbessern.

Bezüglich der Mathematiklastigkeit wurde beim Beispiel „Pi“ die Herleitung der zur verwendenden Formel (Verhältnis der Fläche des Quadrates zur Fläche des Inkreises) eingefügt, so dass eventuell auch auf diese aufsetzend das Beispiel umgesetzt werden kann.

Beim Beispiel „Der einarmige Bandit“ wurden die Wahrscheinlichkeiten aus dem Musterprogramm hinzugefügt, welche verwendet werden können. Dies bedingt dann aber, dass die Anzahl der Symbole pro Walze und die Walzenanzahl nicht geändert werden darf. Diesbezüglich gehen dann also Entscheidungsfreiheiten bei der Ausprogrammierung verloren. Es bietet sich durch diese Vorgehensweise aber auch die Möglichkeit an, die Grundidee dieser Beispielaufgabe mit unterschiedlichen Schwierigkeitsgraden zu versehen. Nach einer Lösung mit den bereits berechneten Vorgaben kann zusätzlich noch durch eine Veränderung der Parameter (Walzenanzahl und/oder Anzahl der Symbole) eine weiterführende, im Schwierigkeitsgrad veränderte, Aufgabenstellung umgesetzt werden.

Von der Ergänzung des Handbuches für Lehrerinnen und Lehrer um Unterlagen für die Schülerinnen und Schüler wurde Abstand genommen. Diese Anregung wurde verworfen was aber nicht der Sinnhaftigkeit der Anregung, sondern einer möglichen Erstellung eines Handbuches für Schülerinnen und Schüler, außerhalb dieser Diplomarbeit, geschuldet ist.

III.2.4 Eigene Reflexion der Rückmeldungen im Rahmen des „Proof of Concept“

Es kann gesagt werden, dass der Aufbau sowie die Strukturierung der Kapitel des Handbuches positiv bewertet wurden. Die Reihenfolge der Kapitel wurde als nachvollziehbar eingeschätzt, die Anzahl der eingefügten Bilder ist passend und diese sind hilfreich und gut gewählt.

Hinsichtlich der Verständlichkeit und der Klarheit des Handbuches sind die Bewertungen ebenfalls im positiven Bereich angesiedelt. Die Erklärungen können als verständlich und von den Schritten her als nachvollziehbar eingestuft werden. Bezüglich der Länge der Erklärungen und der Vollständigkeit der Theorie wurden Verbesserungen angeregt, welche auch umgesetzt wurden.

Die Bewertung der Beispielaufgaben fiel im Prinzip auch positiv aus, es war aber der Punkt, bei welchem die Rückmeldungen am kontroversesten waren. Hinsichtlich der Mathematiklastigkeit habe ich lange mit mir gerungen, ob ich andere Beispielaufgaben, mit weniger Bezug zur Mathematik, verwenden soll. Schlussendlich habe ich diese aber in der ursprünglichen Form belassen. Dies begründet sich primär damit, dass mir eine fächerübergreifende Stoffvermittlung besonders wichtig erscheint und diese im Lehrplan auch dezidiert angeführt ist. Ich gehe auch davon aus, dass die Berechnungen für Schülerinnen und Schüler ab der 10. Schulstufe durchführbar sind.

Allgemein wurde die Vollständigkeit und Exaktheit des Handbuches als Vorteil beschrieben. Es wurde aber auch darauf hingewiesen, dass beim Einstieg in das „Coding“ es auch von Vorteil ist, wenn zunächst bewusst Lücken gelassen werden um ein positives Gefühl zu geben. Die Exaktheit kann auch später gefunden werden. Im Prinzip stimme ich dem zu, eine Überforderung der Schülerinnen und Schüler in der Anfangsphase des „Coding“ ist sicherlich kontraproduktiv. Da es sich aber um ein Lehrerinnen- und Lehrerhandbuch und nicht um ein Schulbuch handelt habe ich daraus aber keinen Handlungsbedarf erkannt. Es bleibt den Lehrerinnen und Lehrern selbst überlassen, ob Sie diese Lücken bei der Wissensvermittlung lassen wollen oder nicht.

Aus der Beantwortung der Kernfragen geht hervor, dass nach Meinung der befragten Lehrerinnen und Lehrer Visual Basic for Applications dazu geeignet ist, um das „Coding“ zu lernen. Es ist auch die Meinung dazu positiv, ob die Lehrerinnen und Lehrer dieses Handbuch im eigenen Unterricht verwenden werden und um es deren Kolleginnen und Kollegen zu empfehlen. Dies spornt mich dazu an das gegenständliche Handbuch für Lehrerinnen und Lehrer, nach und außerhalb dieser Diplomarbeit, weiter zu entwickeln und zu verbessern.

III.2.5 Einschränkungen der gegenständlichen Studie

Trotz aller Gewissenhaftigkeit bei der Erstellung und Auswertung dieser Studie ist diese mit nachstehenden Einschränkungen behaftet:

- Es wurden nur Lehrerinnen und Lehrer in den Bezirken Mattersburg und Eisenstadt in die Studie eingebunden.
- Es ist nur eine geringe Anzahl an Rückmeldungen in diese Studie eingeflossen.
- Das Handbuch wurde nur gelesen und nicht im Unterricht erprobt.

IV. Conclusio, Fazit und Ausblick der Diplomarbeit

Diese Diplomarbeit beschäftigt sich mit der Frage, welche Lerntheorien einerseits und welche Unterrichtsmethoden beziehungsweise didaktische Ansätze andererseits dazu geeignet, um den Einstieg in die Programmierung mit Visual Basic for Applications (VBA) mit Microsoft Excel an Oberstufen in allgemeinbildenden höheren Schulen (AHS) im Rahmen schulautonomen Möglichkeiten der „Neuen Oberstufe“ zu unterrichten. Es wurde für Lehrende ein Handbuch entworfen, welches auf seine Verwendbarkeit im Unterricht überprüft wurde.

Durch die eingehende Beschäftigung mit Visual Basic for Applications mit Excel, dem Lehrplan für allgemeinbildende höhere Schulen, den klassischen Lerntheorien und ausgewählten Unterrichtsmethoden im Zuge dieser Diplomarbeit kann ein fundierter Zusammenhang zum erstellten Lehrerinnen und Lehrerhandbuch erkannt werden.

Visual Basic for Applications mit Excel erfüllt die Voraussetzungen um im Rahmen der schulautonomen Möglichkeiten eingesetzt werden zu können, da neben Programmierkenntnissen auch zusätzliche Kenntnisse über Microsoft Excel vermittelt werden. Es eignet sich um fächerübergreifende Unterrichtseinheiten mit realitätsnahen Beispielen abzuhalten.

Es wurden passende Unterrichtsmethoden für die erforderliche Wissensvermittlung, welche auf den klassischen Lerntheorien basieren, gefunden. Diese ermöglichen es sowohl Basiswissen effektiv zu vermitteln, als auch eine weiterführende Kompetenzaneignung und anschließende Kompetenzumsetzung sicherzustellen.

Aus den Rückmeldungen des „Proof of Concept“ geht hervor, dass einerseits Visual Basic for Applications dazu geeignet ist, um „Coding“ zu lernen, andererseits bekam ich auch die positive Rückmeldung von erfahrenen Lehrerinnen und Lehrern, dass das erstellte Lehrerinnen- und Lehrerhandbuch im Unterricht dazu verwendet werden kann.

Als Fazit kann gesagt werden, dass erfahrene Lehrpersonen denken, dass Visual Basic for Applications mit Excel dazu geeignet ist, um den Einstieg in die Programmierung im

Rahmen der schulautonomen Möglichkeiten der neuen Oberstufe an allgemeinbildenden höheren Schulen zu unterrichten.

Durch die vorangeführten positiven Erkenntnisse angespornt werde ich mich auch künftig mit diesem Thema beschäftigen. Ich werde an der Weiterentwicklung des Lehrerinnen- und Lehrerhandbuches arbeiten und dieses auch selbst im Unterricht einsetzen. Ich überlege auch ein Nachschlagewerk für Schülerinnen und Schüler zu konzipieren.

V. Literaturverzeichnis

BMB Bundesministerium für Bildung. (September 2016). *Die neue Oberstufe – Individuell und kompetenzorientiert, Grundinformation und Ziele im Überblick*. Abgerufen am 11. März 2018 von https://bildung.bmbwf.gv.at/schulen/unterricht/ba/nost/grundinformation.pdf?6aanql

BMBMF Bundesministerium für Bildung, Wissenschaft und Forschung. (8. Jänner 2018). *Die Neue Oberestufe*. Abgerufen am 11. März 2018 von Die Neue Oberestufe: https://bildung.bmbwf.gv.at/schulen/unterricht/ba/nost/index.html

Buhl, A. P., & Strauch , P. P. (2005). *Grundkurs VBA: Einführung in die Programmentwicklung mit Visual Basic for Applications in Excel.* München: Oldenbourg Wissenschaftsverlag GmbH.

Bundesministerium für Digitalisierung und Wirtschaftsstandort. (11. März 2018). *Gesamte Rechtsvorschrift für Lehrpläne – allgemeinbildende höhere Schulen, Fassung vom 11.03.2018*. Abgerufen am 11x. Mätz 2018 von https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10008568

Die Presse. (16. Jänner 2018). Abgerufen am 11. März 2018 von Die Presse: https://diepresse.com/home/bildung/schule/5354598/Neue-Oberstufe-wird-nochmals-verschoben

Drumm, J. (2007). *Methodische Elemente des Unterrichts.* Göttingen: Vandenhoeck & Ruprecht GmbH & Co. KG.

Gudjons, H. (2011). *Frontalunterricht - neu entdeckt.* Bad Heilbrunn: Verlag julius Klinkhardt.

Hölscher, L. (21. 03 2018). *Programmieren zu zweit?* Von http://www.akademie.de/wissen/programmieren-zu-zweit abgerufen

Humbert, L. (2006). *Didaktik der Informatik.* Wiesbaden: B.G. Teubner Verlag.

Khan Academy. (21. März 2018). *Pair programming in the classroom*. Von https://www.khanacademy.org/resources/k-12-teachers-1/teaching-computing/a/pair-programming-in-the-classroom abgerufen

Kofler, M., & Nebelo, R. (2016). *Excel 2016 programmieren: Abläufe automatisieren, (Office-)Add-ins und Anwendungen entwickeln.* München: Carl Hanser Verlag.

Meir, S. (2006). *Didaktischer Hintergrund lerntheorien.* Abgerufen am 26. März 2018 von https://lehrerfortbildung-bw.de/st_digital/elearning/moodle/praxis/einfuehrung/material/2_meir_9-19.pdf

Meyer, H. (2009). *Unterrichtsmethoden.* Berlin: Cornelsen Scriptor.

Meyer, H. (2011). *Was ist guter Unterricht?* Berin: Cornelsen Scritpor.

Riedl , A. (2004). *Grundlagen der Didaktik.* Franz Steiner Verlag Wiesbaden GmbH.

Rogers, C. R. (1983). *Freedom to Learn for the 80's.* Columbus: C.E. Merrill Publishing Company.

Schaub, H., & Zenke, K. G. (2000). *Wörterbuch Pädagogik.* München: dtv - deutscher Taschenbuch Verlag.

Schlimok, J. (2015). *Einzelarbeit oder Stillarbeit – Voraussetzung für andere Sozialformen im Unterricht*. Abgerufen am 22. März 2018 von https://www.forrefs.de/grundschule/unterricht/unterricht-halten/organisationsformen/einzelarbeit-oder-stillarbeit-voraussetzung-fuer-andere-sozialformen-im-unterricht.html

Schlimok, J. (2015). *Frontalunterricht – besser als sein Ruf*. Abgerufen am 19. März 2018 von https://www.forrefs.de/grundschule/unterricht/unterricht-halten/organisationsformen/frontalunterricht-besser-als-sein-ruf.html

Stangl, W. (2018). *Online Lexikon für Psychologie und Pädagogik*. Abgerufen am 26. März 2018 von http://lexikon.stangl.eu/8649/lerntheorie/

Theis , T. (2016). *Einstieg in VBA mit EXCEL* (4. aktualisierte Aufllage 2016 Ausg.). Bonn: Rheinwerk Verlag GmbH.

Tulodziecki, G., Herzig, B., & Blömeke, S. (2017). *Gestaltung von Unterricht.* Bad Heilbrunn: Verlag Julius Klinkhardt.

Walkenbach, J. (2016). *Excel-VBA für Dummies.* (J. Muhr, Übers.) Weinheim, Deutschland: WILEY-VCH Verlag GmbH & Co KG.

Wiechmann , J., & Wildhirt, S. (2016). *12 Unterrichtsmethoden.* Weinheim: Beltz Verlag.

Wiechmann, J. (2002). Methodenvielfalt für die Schulpraxis. *Pädagogische Rundschau* , S. 393-409.

VI. Anhang

VI.1 Anhang 1: Fragebogen „Proof of Concept“

Fragebogen zum Lehrerinnen- und Lehrerhandbuch für den Einstieg in die Programmierung mit Visual Basic for Applications (VBA) mit Microsoft Excel

Sehr geehrte Lehrerinnen, sehr geehrte Lehrer!

Das Lehrerinnen- und Lehrerhandbuch habe ich im Rahmen meiner Diplomarbeit verfasst. Die Diplomarbeit beschäftigt sich mit der Frage welche Lerntheorien, Unterrichtsmethoden beziehungsweise didaktischen Ansätze sich besonders gut dafür eignen, um den Einstieg in die Programmierung mit Visual Basic for Applications (VBA) mit Microsoft Excel an Oberstufen in allgemeinbildenden höheren Schulen (AHS), im Rahmen der schulautonomen Möglichkeiten der „Neuen Oberstufe“, zu unterrichten.

Das „Proof of Concept“ meiner Ausarbeitung erfolgt durch Ihre Begutachtung. Ich bitte Sie mir Ihr Feedback durch die Befüllung des nachstehenden Fragebogens zu geben. Ihre Anregungen, Ideen und Verbesserungsvorschläge werde ich in die Endversion des Handbuches einarbeiten.

Vielen Dank im Voraus, dass Sie sich die dafür Zeit nehmen um mein Lehrerinnen- und Lehrerhandbuch zu begutachten und den Fragebogen auszufüllen.

	stimme nicht zu	stimme eher nicht zu	stimme eher zu	stimme zu	keine Antwort
Aufbau und Struktur des Handbuches					
Dieses Handbuch ist übersichtlich aufgebaut	☐	☐	☐	☐	☐
Die Kapitel sind gut strukturiert	☐	☐	☐	☐	☐
Die Reihenfolge der Themen ist nachvollziehbar gestaltet	☐	☐	☐	☐	☐
Die eingefügten Bilder sind hilfreich und gut gewählt	☐	☐	☐	☐	☐
Die Anzahl der Bilder ist passend	☐	☐	☐	☐	☐

Verbesserungsvorschläge:

	stimme nicht zu	stimme eher nicht zu	stimme eher zu	stimme zu	keine Antwort
Verständnis und Klarheit des Handbuches					
Die Erklärungen sind verständlich formuliert	☐	☐	☐	☐	☐
Die Länge der Erklärungen ist passend	☐	☐	☐	☐	☐
Die einzelnen Schritte der Erklärungen sind nachvollziehbar	☐	☐	☐	☐	☐
Das Wesentliche der Theorie ist vorhanden	☐	☐	☐	☐	☐

Verbesserungsvorschläge:

Beispielaufgaben					
Die Anzahl der Beispiele ist gut gewählt	☐	☐	☐	☐	☐
Der Schwierigkeitsgrad der Beispiele ist angemessen	☐	☐	☐	☐	☐
Die Erklärungen der Beispiele sind verständlich formuliert	☐	☐	☐	☐	☐
Die Beispielaufgaben sind fächerübergreifend verwendbar	☐	☐	☐	☐	☐
Die Beispielaufgaben sind realitätsnah	☐	☐	☐	☐	☐

Verbesserungsvorschläge:

Kernfragen

Finden Sie VBA geeignet um „Coding“ zu lernen?	☐	☐	☐	☐	☐
Können Sie sich vorstellen, dieses Handbuch in Ihrem Unterricht einzusetzen?	☐	☐	☐	☐	☐
Würden Sie dieses Handbuch ihren KollegInnen weiterempfehlen?	☐	☐	☐	☐	☐

Angaben zur Person

Name _____

Wie lange sind Sie als Lehrerin bzw. Lehrer tätig? _____

In welchem Schultyp unterrichten Sie? _____

Welche Fächer unterrichten Sie? _____

Haben Sie schon einmal „Coding“ unterrichtet? JA ☐ NEIN ☐

Wie gut kennen Sie sich mit VBA aus? 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Wenn Sie VBA kennen, haben Sie es schon unterrichtet? JA ☐ NEIN ☐

Offene Frage

Was Sie mir sonst noch sagen möchten bzw. sonstige Tipps/Hinweise:

VI.2 Anhang 2: Handbuch für Lehrerinnen- und Lehrer

Das Lehrerinnen- und Lehrerhandbuch finden im Folgenden, also ab dem nächsten Blatt. Da dieses Handbuch als selbstständiges Werk, also losgelöst von dieser Diplomarbeit, verwendet werden soll, weist dieses alle Merkmale eines eigenständigen Buches auf und hat somit zum Beispiel ein eigenes Inhalts- und Quellenverzeichnis oder eine eigene Seitennummerierung.

Handbuch

für Lehrerinnen und Lehrer

für den Einstieg in die Programmierung mit

Visual Basic for Applications (VBA)
mit Microsoft Excel

Inhalt

Vorwort

Das gegenständliche Lehrerinnen- bzw. Lehrerhandbuch habe ich im Rahmen meiner Diplomarbeit verfasst. Die Diplomarbeit beschäftigt sich mit der Frage welche Lerntheorien, Unterrichtsmethoden bzw. didaktischen Ansätze sich besonders gut dafür eignen, um den Einstieg in die Programmierung mit Visual Basic for Applications (VBA) mit Microsoft Excel (Excel) an Oberstufen in allgemeinbildenden höheren Schulen (AHS) im Rahmen schulautonomer Möglichkeiten der „Neuen Oberstufe“(NOST) zu unterrichten.

Das Handbuch soll Lehrende dabei unterstützen Schülerinnen und Schülern in einem Einführungsunterricht Basiskenntnisse in der Programmierung, dem sogenannten „Coding“, zu vermitteln. Lehrerinnen und Lehrer, welche mit Visual Basic for Applications (VBA) mit Excel nicht vertraut sind, sollen dadurch einen groben Überblick über die zur Verfügung stehenden Möglichkeiten erlangen. Es soll diesen einen schnellen Einstieg in die Materie ermöglichen und auch als Nachschlagewerk und zur Ideenfindung dienen.

Weiters sollen dadurch Lernende dazu angeregt und motiviert werden, sich selbstständig mit der Programmierung auseinanderzusetzen. Durch die Verwendung dieses in Microsoft Excel integrierten Tools haben viele Schülerinnen und Schüler die Möglichkeit, ohne langwierige und mitunter komplexe Installationen, sich mit einer erprobten und modernen objekt- und ereignisorientierten Programmiersprache zu beschäftigen.

Das Handbuch erhebt keinen Anspruch auf Vollständigkeit und ist wirklich nur für die Einführung in Programmierung gedacht. Visual Basic for Applications (VBA) bietet viele weitere Möglichkeiten, insbesondere in der Interaktion zwischen Microsoft Excel (Excel) und VBA-Code, welche nicht behandelt werden, da diese über eine Einführung weit hinausgehen. Viele Einsteiger würden wahrscheinlich, zumindest anfänglich, überfordert werden und möglicherweise die Lust auf mehr verlieren.

Das gegenständliche Handbuch wurde unter Verwendung von Microsoft Excel 2016 erstellt. Eine Kompatibilität zu niedrigeren Versionen, insbesondere Microsoft Excel 2013, sollte gegeben sein. Eventuell sind bei der Verwendung von älteren Versionen die in diesem Handbuch gezeigten Abbildungen oder getätigten Beschreibungen nicht mit den dort sichtbaren Benutzeroberflächen ident.

Es wurden von mir diverse Erläuterungsprogramm sowie Musterprogramm zu den Musteraufgaben erstellt. Die erstellten Programme stehen unter dem Link https://drive.google.com/open?id=1P7nvdq1DleY0XaX3NtdCG1OUneiOaQua zum Download und zur allgemeinen freien Verwendung zur Verfügung.

Da ich dieses Handbuch in der Folge erweitern und eventuell auch um ein Nachschlagewerk für Schülerinnen und Schüler erweitern möchte bin ich für Anregungen oder Hinweise dankbar. Ebenso stehe ich gerne für allfällige Rückfragen unter der E-Mailadresse piiinqq@gmail.com zur Verfügung.

In diesem Handbuch gezeigte Abbildungen ohne explizite Quellenangabe sind von mir selbst erstellte Screenshots.

Ich wünsche allen Benutzerinnen und Benutzern dieses Handbuchs ein gutes Gelingen und viel Spaß beim Programmieren.

Was ist Visual Basic for Applications (VBA)?

Visual Basic for Applications (VBA) ist eine von Microsoft entwickelte Programmiersprache welche einen integrierenden Bestandteil der Microsoft Office Produkte wie zum Beispiel Excel darstellt. Es können damit einerseits die Microsoft Office (MS-Office) Anwendungen gesteuert werden, andererseits kann man damit auch eigenständige Programme erstellen, entweder mit oder ohne Verwendung von Excel-Komponenten. Die Programmiersprache "Visual Basic for Applications" ist von "Visual Basic" abgeleitet. Die zu verwendende Syntax entspricht jener von Visual Basic. (vgl. Walkenbach, 2016, S. 33)

Visual Basic for Applications ist eine Makrosprache, welche nur innerhalb einer anderen Applikation ausgeführt werden kann, in diesem Fall eben in Microsoft Office (MS-Offie) Anwendungen. Visual Basic for Applications (VBA) ist eine vollwertige Programmiersprache, in welcher wichtige Konzepte moderner Programmiersprachen wie zum Beispiel Datentypen, dynamische Felder oder rekursive Funktionen zur Verfügung stehen. VBA ist objektorientiert, wodurch eine effektive Programmierung möglich wird. Als Objekte stehen zum Beispiel Zellen, Zellbereiche, Tabellenblätter etc. zur Verfügung. Als ereignisorientierte Programmiersprache reagiert VBA auf Ereignisse, wie zum Beispiel auf einen Klick auf eine Schaltfläche, mit welcher ein Programmcode verknüpft werden kann. VBA ist komponentenorientiert was bedeutet, dass auf Lösungen bereits bestehender Programme (Komponenten) zurückgegriffen werden kann. (vgl. Buhl & Strauch , 2005, S. 2f.)

Warum Visual Basic for Applications (VBA)?

Es sprechen mehrere Fakten dafür, warum ich mich für VBA entschieden habe. Der Grund liegt in einem darin, dass bereits heute in den AHS-Oberstufen die MS-Office Anwendungen unterrichtet werden. Die Schülerinnen und Schüler lernen dabei wie die einzelnen Komponenten des MS-Office-Paketes funktionieren und eingesetzt werden können. Mit VBA können die Möglichkeiten der Verwendung erweitert sowie Abläufe automatisiert und damit sehr beschleunigt werden. Da die MS-Office Anwendungen in vielen Unternehmen und auch im öffentlichen Bereich eingesetzt werden, können diesbezügliche Zusatzkenntnisse der Schülerinnen und Schüler für diese nur von Vorteil sein.

Ein weiterer Vorteil liegt darin, dass VBA als Komponente von MS-Office mit der bloßen Installation der MS-Office Anwendungen automatisch installiert ist. Es kann also ohne weiteren Aufwand einer Softwareinstallation mit dem Unterricht und dem Erlernen des Programmierens begonnen werden. Es fallen auch keine weiteren Kosten wie zum Beispiel Lizenzkosten an.

VBA ist eine vollwertige Programmiersprache und ist in seinen Sprachelementen mit anderen objekt- und ereignisorientierten Programmiersprachen vergleichbar, sodass ein späterer Umstieg auf andere moderne Programmiersprachen, zum Beispiel auf C++ oder Visual Basic.Net, leichter fallen sollte.

Aufbau dieses Handbuches

Im ersten Teil finden Sie Informationen über diverse Einstellungen und Parametrierungen, welche Ihnen einen sinnvollen Umgang mit VBA ermöglichen. Weiters erfahren Sie, wie man die Entwicklungsumgebung von VBA, den „*Visual Basic Editor*“ (VBE), aufrufen kann und finden Erklärungen über die Bestandteile der VBE-Oberfläche.

Im nächsten Teil führt Sie dieses Handbuch durch die ersten Interaktionen mit VBA. Sie werden Daten eingeben und entsprechende Ausgaben erhalten. In der Folge erfahren Sie etwas über Methoden, Eigenschaften und Ereignisse von Objekten, die Objekthierarchie und die Verwendung von Variablen, Arrays, Prozeduren und Funktionen. Es stehen Ihnen dabei Excelfiles zur Verfügung, in welchen die Themen veranschaulicht sind.

Der dritte Teil beschäftigt sich mit den vorhandenen Operatoren und den wichtigsten Sprachelementen von VBA. Sie erfahren etwas über Entscheidungsbefehle (IF oder Select Case) und über die Funktionsweisen von Schleifen (For-Schleifen und Do-Schleifen).

Es wird empfohlen die Teile eins bis drei mit den Schülerinnen und Schülern durchzugehen, bevor die Musteraufgaben des vierten Teiles abgearbeitet werden. Es sollten sodann alle relevanten Informationen für eine Ausprogrammierung gegeben sein.

Im vierten Teil finden Sie Musteraufgaben samt Musterlösungen, welche in der vorliegenden Form, oder auch in Abwandlungen, umgesetzt werden können. Mit Hilfe der Musterlösungen kann den Schülerinnen und Schülern anschaulich gezeigt werden, was programmiert werden soll. Der Mustercode bietet auch die Möglichkeit einen gangbaren Ansatz der Ausprogrammierung zu präsentieren. Bei den Musterlösungen wurde darauf Wert gelegt, dass der Programmcode für Programmiereinsteigerinnen und Programmiereinsteiger nachvollziehbar bleibt. Es wurde darauf Bedacht genommen, dass Wissen an Neueinsteigerinnen und Neueinsteiger im Erstellen von Programmen zu vermitteln ist.

Vorbereitende Arbeiten

Die Standardeinstellungen von Microsoft Excel, welche bei der Installation gesetzt werden, müssen geändert werden. Mit der Änderung der Sicherheitsstufe und nach der Einblendung der „Entwicklertools“ kann der „*Visual Basic Editor*“ (VBE) genutzt werden.

Änderung der Sicherheitsstufe

Aus Sicherheitsgründen wurde seit der Version Excel 2007 die Verwendung von Makros erschwert. Mit der Installation wird automatisch die höchste Sicherheitsstufe aktiviert, wodurch die Verwendung von Makros unterbunden wird. Diese Einstellung muss nur einmalig geändert werden und bleibt erhalten. (vgl. Held, 2016, S. 1f.)

Gehen Sie dabei wie folgt vor:

1. Microsoft Excel starten.
2. Im Menüband „*Datei*“ anklicken.
3. Den Befehl „*Optionen*“ wählen.
4. Wählen Sie im Dialog „*Excel-Optionen*“ die Rubrik „*Trust Center*“ aus.
5. Klicken Sie auf den Button „*Einstellungen für das Trust Center...*“.
6. Klicken Sie auf die „*Makroeinstellungen*“.
7. Wählen Sie die Option „*Alle Makros aktivieren*“.
8. Aktivieren Sie das Kontrollkästchen bei „Zugriff auf das VBA-Projekt-Objekt vertrauen“.
9. Bestätigen Sie mit „OK“.

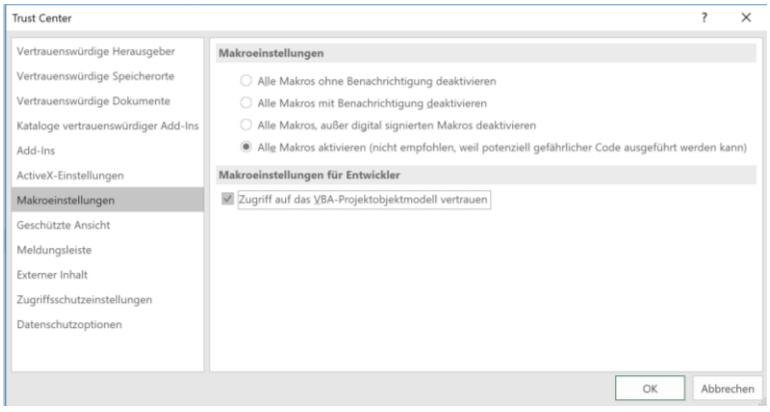


Abbildung 1: Anpassung der Makroeinstellungen und Zustimmung zum Zugriff auf das VBA-Objektmodell.

Einblendung der Entwicklertools

Es steht in der Oberfläche von Excel eine eigene Registerkarte mit dem Namen „*Entwicklertools*“ für die Verwaltung und Programmierung von Makros zur Verfügung, welche den Standardanwendern jedoch verborgen ist. So können zum Beispiel Schaltflächen in Tabellen eingefügt und diesen Makros zugewiesen werden. Es besteht aber auch die Möglichkeit zur Aufzeichnung von Makros. (vgl. Held, 2016, S. 2f.)

Gehen Sie dabei wie folgt vor:

1. Im Menüband „*Datei*“ anklicken.
2. Den Befehl „*Optionen*“ wählen.
3. Wählen Sie im Dialog „*Excel-Optionen*“ die Rubrik „*Menüband anpassen*“ aus.
4. Aktivieren Sie das Kontrollkästchen „*Entwicklertools*“ im Feld „*Hauptregisterkarten*“
5. Bestätigen Sie mit „*OK*“.

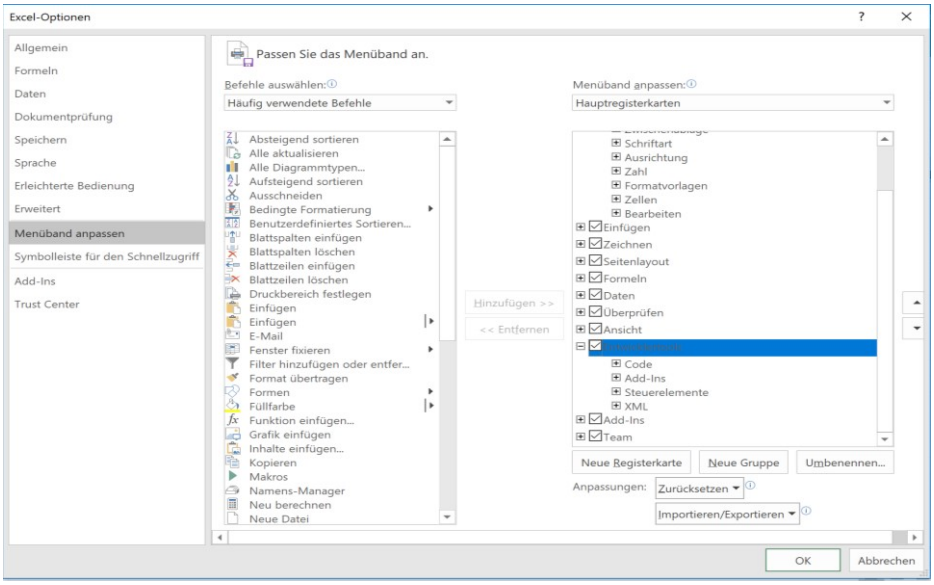


Abbildung 2: Einblendung des Werkzeuges "Entwicklertools".

Nunmehr steht das Menüband „*Entwicklertools*“ zur Verfügung und kann verwendet werden. Die Vorbereitungen sind abgeschlossen.

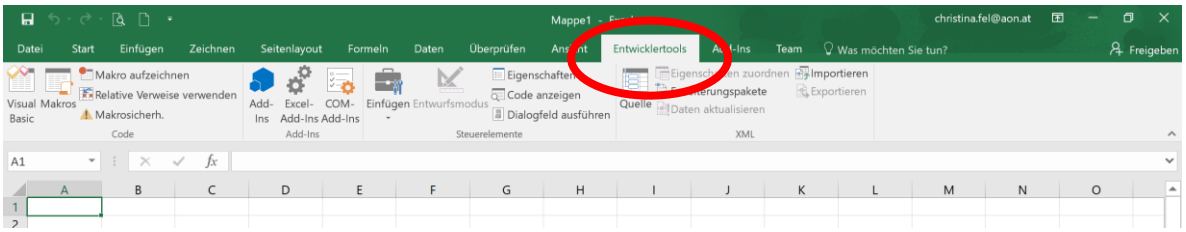


Abbildung 3: Register "Entwicklertools".

Aufzeichnung von Makros

Ein Makro besteht aus einer Reihe von Anweisungen, welche nacheinander abgearbeitet werden. Es wird dadurch ein bestimmter Vorgang in Excel ausgeführt. Diese Anweisungen sind in der Programmiersprache „*Visual Basic for Applications*“ (VBA) geschrieben. VBA-Programme können aber nicht nur einfache Aufgaben erledigen, sie können darüber weit hinausgehen und auch sehr komplexe Abläufe steuern. (vgl. Theis , 2016, S. 20)

Als erstes Beispiel werden wir einen in der Zelle „*A1*“ befindlichen Text „Hallo“ aus dieser in die Zelle „*C1*“ verschieben und diesen von der Farbe „Schwarz“ auf „Rot“ ändern.

Gehen sie dabei wie folgt vor:

1. Schreiben Sie den Text „Hallo“ in die Zelle „*A1*“.
2. Drücken Sie auf die Schaltfläche „*Makro aufzeichnen*“
3. Belassen Sie den Makronamen „*Makro1*“ unverändert oder ändern Sie diesen zum Beispiel auf „*vonA1nachC1*“.
4. Bestätigen Sie mit dem Button „*OK*“ und es werden sodann alle folgenden Aktivitäten, welche Sie in Excel durchführen (z.B. mit der Maus oder auch über die Tastatur), aufgezeichnet.
5. Klicken Sie in die Zelle „*A1*“
6. Schneiden Sie den Inhalt mit der Tastenkombination [Strg] + [X] aus
7. Klicken Sie in die Zelle „*C1“*
8. Fügen Sie den vorher ausgeschnittenen Inhalt mit der Tastenkombination [Strg] + [V] ein.
9. Ändern Sie die Textfarbe auf „ROT“.
10. Wechseln Sie zu den „*Entwicklertools*“ und drücken sie die Schaltfläche „*Aufzeichnung beenden*“.

Die Aufzeichnung wurde jetzt beendet und Sie können jetzt dem im Makro generierten VBA-Programmcode im „*Visual Basic Editor*“ ansehen und bearbeiten. Drücken Sie im Menü „*Entwicklertools*“ auf die Schaltfläche „*Makros*“ und drucken Sie den Button „*Bearbeiten*“.

Sie sehen nun zum ersten Mal den „*Visual Basic Editor*“ (VBE). Die einzelnen Bestandteile werden im nachfolgenden Abschnitt „*Visual Basic Editor (VBE) - Die* Entwicklungs-umgebung“ dieses Handbuches erläutert.

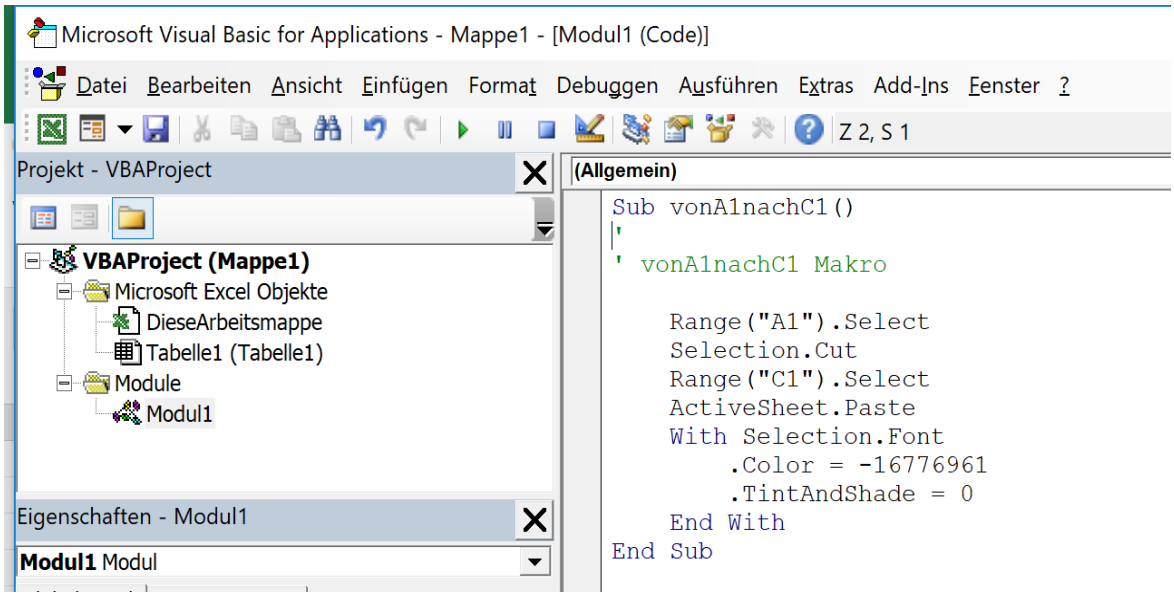


Abbildung 4: Generierter VBA-Code des Makros "vonA1nachC1"

Wie kann dieser Code jetzt interpretiert werden? Wir wissen noch von vorher, dass wir den Text „Hallo“ von der Zelle „*A1*“ ausschneiden und in der Zelle „*C1*“ einfügen wollen, gleichzeitig sollte die Farbe auf „Rot“ geändert werden.

Der Programmcode des Makros „*vonA1nachC1*“ ist in einer Sub-Prozedur zwischen den Anweisungen „*Sub*“ und „*End Sub*“ eingebettet. Zwischen den nachgestellten Klammern können Variablen in die Prozedur übergeben werden (näheres dazu später).

Die Darstellung in der Farbe „Blau“ zeigt immer den Anfang und das Ende eines Blocks an. Um die Übersichtlichkeit zu bewahren, werden Programmzeilen eines Blockes immer gleich eingerückt. So kann bei komplexen Programmen die Übersicht leichter bewahrt werden.

Mit Hilfe des Zeichens „*Apostrophe*“ (') wird ein Kommentar gekennzeichnet, welcher beim Programmablauf nicht abgearbeitet wird. Die ganze Zeile wird in der Farbe „Grün“ dargestellt.

```
Sub vonA1nachC1()                => Beginn der Sub-Prozedur „vonA1nachC1“
' vonA1nachC1 Makro              => eine Kommentarzeile
  Range("A1").Select             => der Bereich A1 wird gewählt
  Selection.Cut                  => der Inhalt wird ausgeschnitten (in
                                    Zwischenablage)
  Range("C1").Select             => der Bereich C1 wird gewählt
  ActiveSheet.Paste              => der Inhalt wird eingefügt (aus Zwischenablage)
  With Selection.Font            => Das Fenster „Schriftart“ wird geöffnet
       .Color = -16776961        => Die Farbe „Rot“ wird gewählt
       .TintAndShade = 0         => Der Text wird in der gewählten Farbe dargestellt
  End With                       => Das Fenster „Schriftart“ wird geschlossen
End Sub                          => Ende der Sub-Prozedur
```

Schreiben Sie nun einen beliebigen Text in die Zelle „A1“, zum Beispiel Ihren Namen. Ändern Sie die Farbe der Zelle „*C1*“ wieder auf Schwarz. Drücken Sie nun in der Menüleiste unter „*Entwicklertools*“ die Schaltfläche „*Makros*“. Es erscheint das Fenster „Makro“. Drücken Sie nun die Schaltfläche „*Ausführen*“.

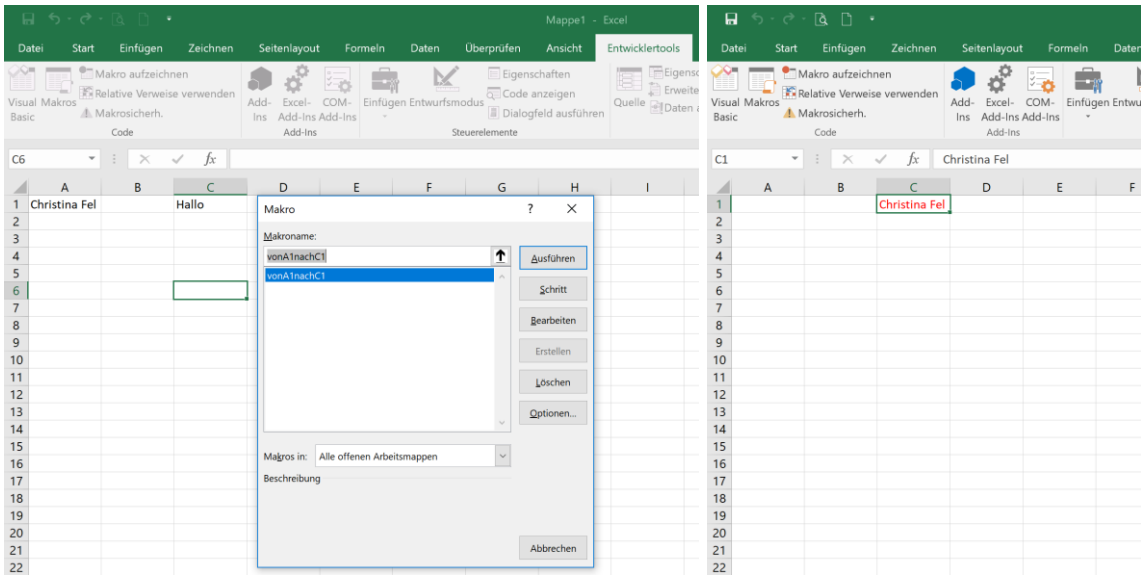


Abbildung 5: Excel Oberfläche vor und nach Ausführung des Makros "vonA1nachC1"

Das erste in VBA erstellte Programm schneidet somit den in der Zelle „*A1*“ gespeicherten Text aus und fügt diesen in der Zelle „*C1*“ wieder ein und färbt diesen rot.

Speichern von Excel-Dateien mit Makros

Seit Excel 2007 können Dateien sowohl mit als auch ohne Makro gespeichert werden. (vgl. Theis , 2016, S. 25) In der benutzten Version von Excel 2016 klicken Sie im Menü „*Datei*“ auf den Punkt „*Speichern unter*“, dann wählen Sie den Pfad.

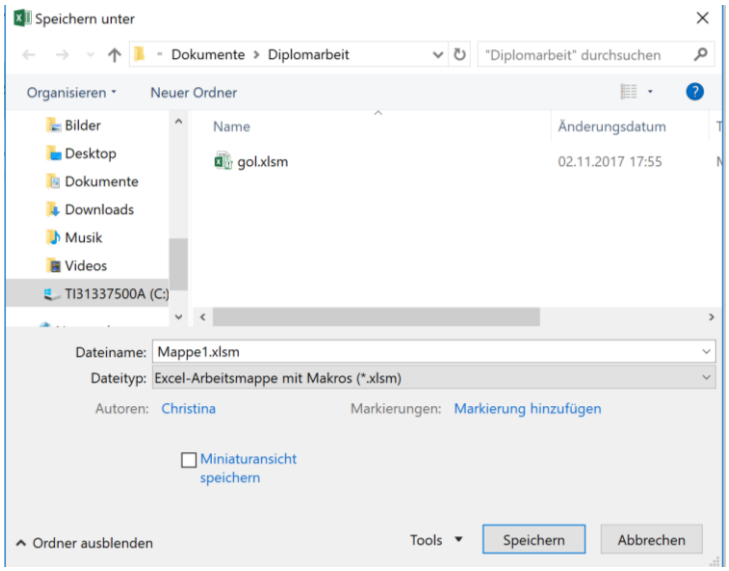


Abbildung 6: Als Arbeitsmappe mit Makros speichern

Ändern Sie den Dateityp auf „*Excel-Arbeitsmappe mit Makros (*.xlsm)*“ und betätigen Sie den Button „*Speichern*“.

Öffnen von Excel-Dateien mit Makros

Beim Öffnen von Excel Dateien welche im *“*.xlsm“* Format gespeichert wurden, erscheint ein standardmäßiger Hinweis *„! SICHERHEITSWARNUNG Makros wurden deaktiviert“*.

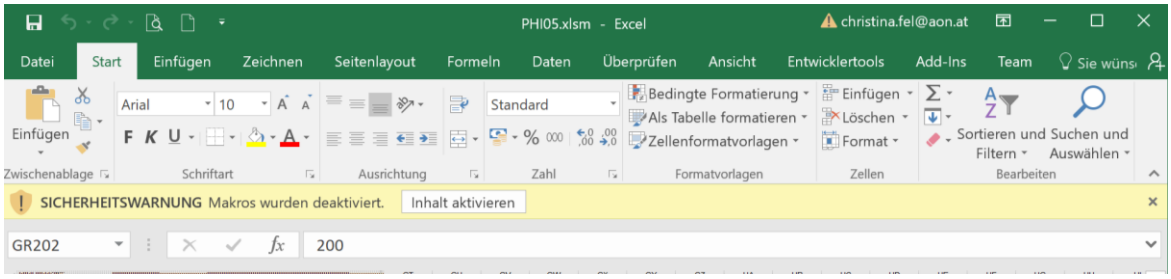


Abbildung 7: Sicherheitswarnung beim Öffnen von Dateien mit Makros

Erst nach dem Anklicken der Schaltfläche „*Inhalt aktivieren*“ können die Makros ausgeführt werden.

Visual Basic Editor (VBE) - Die Entwicklungsumgebung

Der Visual Basic Editor (VBE) als Entwicklungsumgebung dient dazu, um den Visual Basic for Applications (VBA) Programmcode zu schreiben. Sie gelangen über die Registerkarte „*Entwicklertools*“ und die Schaltfläche „*Visual Basic*“ oder über die Tastenkombination [*Alt*] + [*F11*] zum Visual Basic Editor. Über diese Tastenkombination gelangen Sie auch wieder zurück zur gewohnten Excel-Oberfläche. (vgl. Theis , 2016, S. 31)

Sie gelangen somit auf die Ihnen teilweise bereits bekannte Oberfläche (Abbildung 4: Generierter VBA-Code des Makros "*vonA1nachC1*")

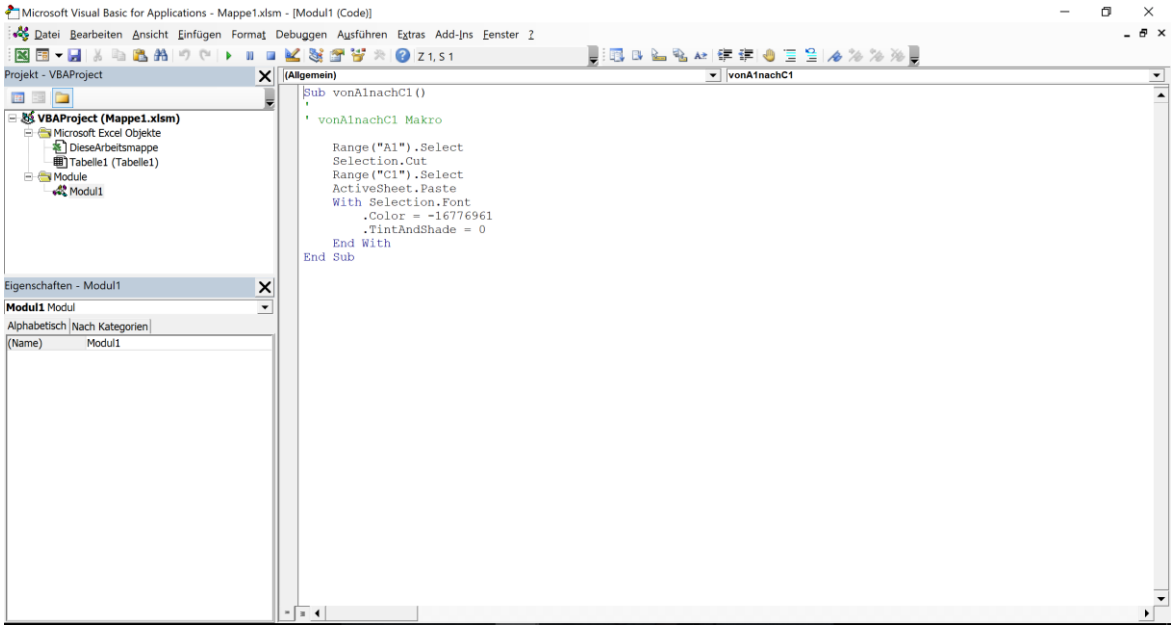


Abbildung 8: VBA Entwicklungsumgebung Visual Basic Editor (VBE)

Es werden also auch die Makros als VBA-Programmcode generiert. Dies ist insbesondere in der Anfangsphase der Programmierung oft sehr hilfreich. So können zum Beispiel fehlende Kenntnisse über Schlüsselwörter relativ einfach kompensiert werden. Man erstellt einfach Makros über die gewünschten Vorgänge und passt diese entsprechend der eigenen Erfordernisse an.

Die Bereiche des Visual Basic Editor (VBE)

Der Visual Basic Editor (VBE) gliedert sich in 6 Bereiche, wobei sich die Oberfläche wie folgt darstellt.

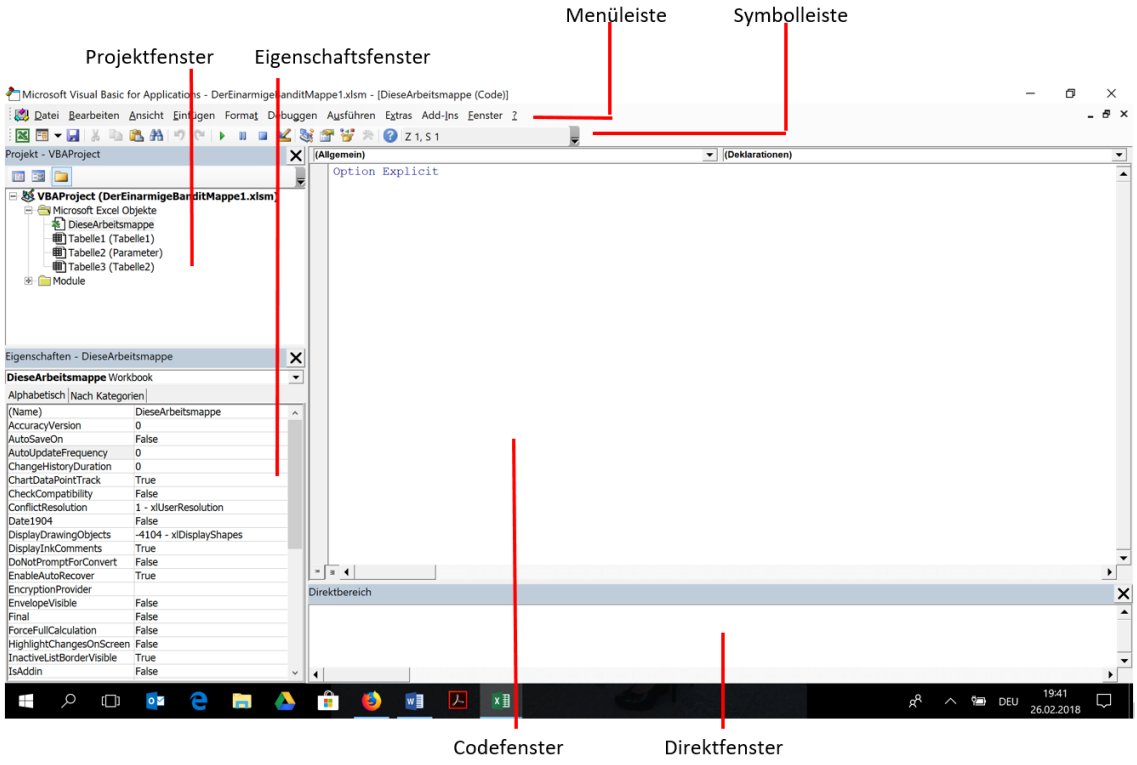


Abbildung 9: Die Bestandteile des VBE

Die Menüleiste

Die Menüleiste enthält Befehle, mit welchen verschiedene Komponenten im VBE manipuliert werden können. Viele der Befehle sind auch mit Tastenkürzel hinterlegt. Eine Vielzahl der Befehle kann aber auch über Kontextmenüs (Klick auf rechte Maustaste) aufgerufen werden. (vgl. Walkenbach, 2016, S. 58f.)

Die Symbolleiste

Es können vier VBE-Symbolleisten eingeblendet werden. Dies erfolgt über die Befehle „*Ansicht*“ und „*Symbolleiste*“. Üblicherweise findet man aber das Auslangen mit der oben dargestellten Standardeinstellung. (vgl. Walkenbach, 2016, S. 59)

Der „Projekt-Explorer“ und das „Eigenschaftsfenster“

Im Projektexplorer werden alle geöffneten Arbeitsmappen sowie die darin enthaltenen Tabellen angezeigt. Mit den beiden linken Schaltflächen der „*Symbolleiste*“ kann jeweils zwischen dem Objekt bzw. dem Code gewechselt werden. Der Projektexplorer steht im engen Konnex zum darunterliegenden Eigenschaftsfenster. Klickt man zum Beispiel im Projekt-Explorer auf das Objekt „*Tabelle1*“ so werden im Eigenschaftsfenster die dazugehörenden Eigenschaften angezeigt. Generell können die Eigenschaften über das Eigenschaftsfenster oder über VBA-Programmcode eingestellt werden. Einige Änderungen über das Eigenschaftsfenster (z.B. *ScrollArea*) bleiben jedoch nicht dauerhaft erhalten. Es empfiehlt sich also die Werte über den VBA-Programmcode beim Eintritt bestimmter Ereignisse, zum Beispiel bei der Aktivierung eines Tabellenblattes, anzupassen. (vgl. Held, 2016, S. 4ff.)

Diverse Änderungen dieser Einstellungen wie zum Beispiel das Verstecken von Tabellenblättern oder die Einschränkung der zugänglichen Bereiche innerhalb eines Tabellenblattes machen bei der Entwicklung professioneller Programme durchaus Sinn und sind auch von Nöten. Es wird aber in der Folge nicht weiter darauf eingegangen, da dies über eine Einführung in die VBA-Programmierung weit hinausgehen würde.

Das Codefenster

Das *„Codefenster“* befindet sich rechts neben dem „*Projekt-Explorer“* und wird nach dem Anlegen eines Moduls sichtbar. Ein Modul entspricht dabei einem Ordner, in welchem in der Folge die Makros (VBA-Programmcode) abgelegt bzw. gespeichert werden. Wichtig dabei ist, dass Tabellen, Module und Makros mit sprechenden Namen bezeichnet werden. Der geschriebene VBA-Programmcode wird dadurch leichter nachvollziehbar, für einen selbst, aber auch für Dritte. (vgl. Held, 2016, S. 10f.)

Bei der Benennung von Makros ist zu beachten, dass

- das erste Zeichen alphanumerisch sein muss,
- kein Leerzeichen enthalten sein darf,
- keine Sonderzeichen (/, %, -, $, [,], ! oder dergleichen) enthalten sein dürfen und
- diese mit einem runden Klammernpaar „( )“ zu beenden sind. (vgl. Held, 2016, S. 11)

Ein gültiger Makroname wäre somit *„ErstesMakro“*

Nunmehr wird das erste Makro selbst angelegt. Folgen Sie dabei bitte den nachstehenden Schritten:

Wählen Sie in der VBE-Umgebung im Menü „*Einfügen*“ den Befehl „*Modul*“ aus.

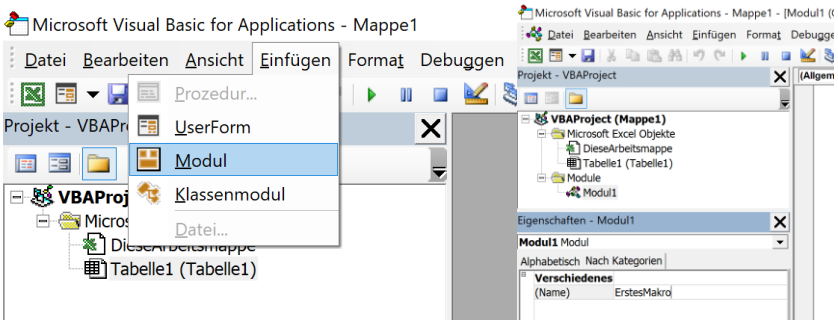


Abbildung 10: Einfügen und Umbenennen eines Moduls

Ändern Sie den Namen des Moduls, indem Sie das angelegte „*Modul1*“ anklicken und danach im darunterliegenden *„Eigenschaftsfenster“* den Namen auf „*ErstesMakro*“ ändern.

Sie sehen im „*Codefenster*“ den Eingabebereich. Ein Makro beginnt mit dem Schlüsselwort „*Sub*“, gefolgt von dem Makronamen (Modulnamen) sowie einem Klammernpaar „*()*“und endet mit Schlüsselwort „*End Sub*“.

Dazwischen muss der jeweilige VBA-Programmcode angeführt werden.

Nachstehend ein Beispiel, in welchem Systemvariablen (Environ) in einer Mitteilungsbox ausgegeben werden.

```
Sub ErstesMakro()

MsgBox Environ("Computername") & " " & Environ("Username") &
vbNewLine & Environ("OS") & " " & Environ(17)

End Sub
```

Um das Makro zu starten drücken Sie „F5“ oder Schaltfläche mit dem grünen Dreieck in der „Symbolleiste“. Markieren Sie das auszuführende Makro und drücken Sie die Schaltfläche „*Ausführen*“.

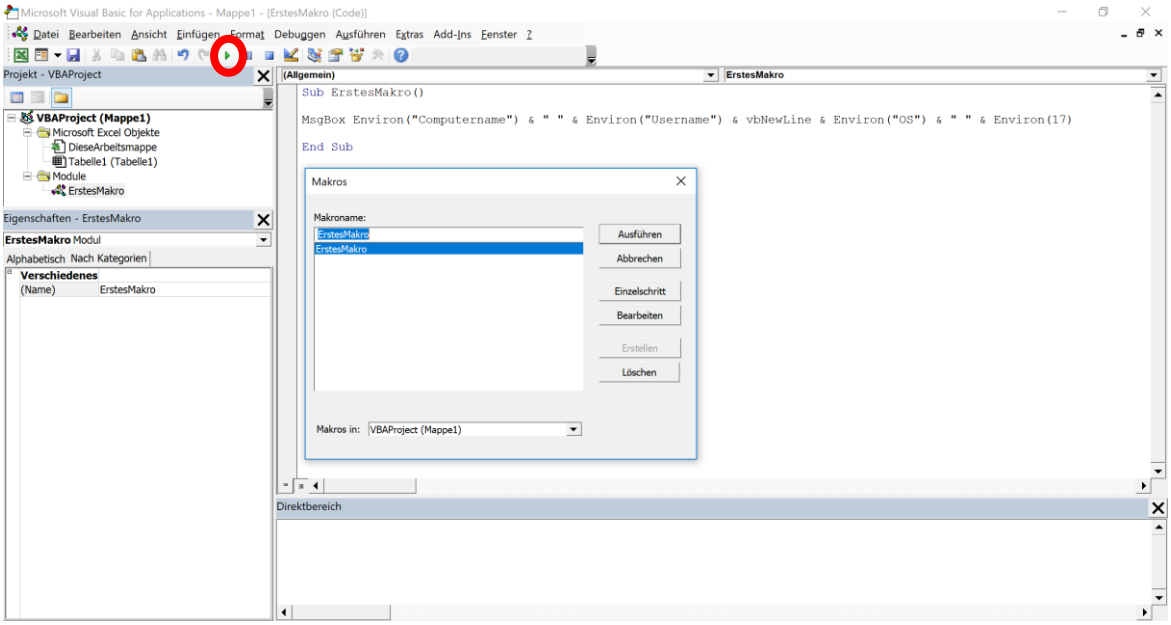


Abbildung 11: "ErstesMakro" ausführen

Das Makro wird ausgeführt und folgende Ausgabe generiert:

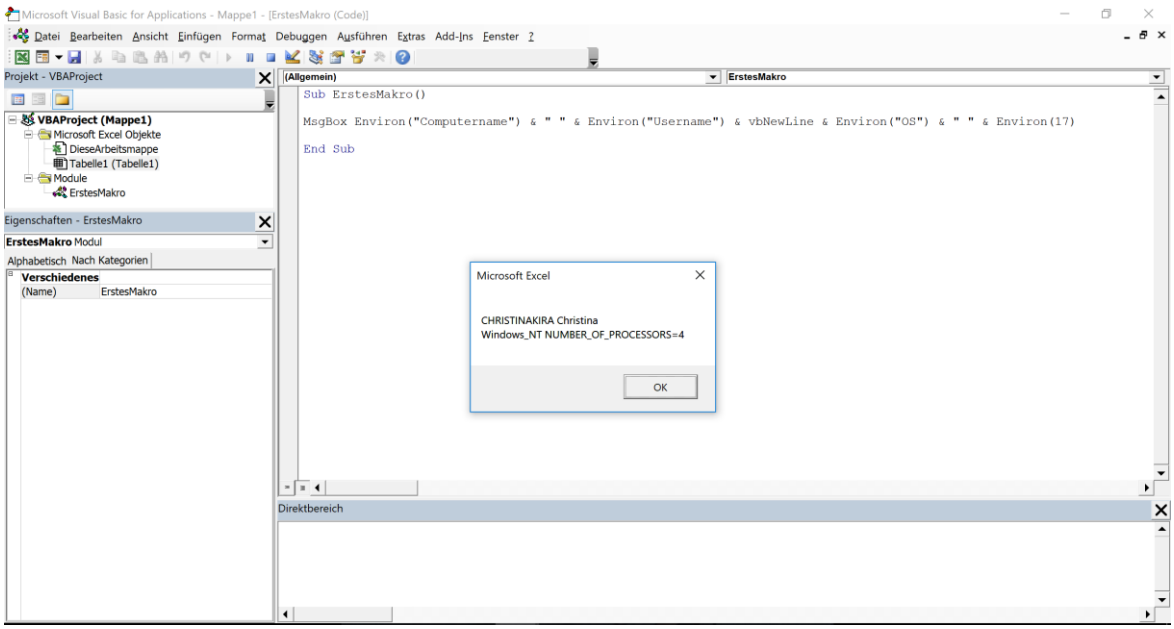


Abbildung 12: Ausgabe "ErstesMakro"

Ausgegeben wird der Computername „CHRISTINAKIRA“, der Username „Christina“ und nach einem Zeilenumbruch das Betriebssystem „Windows_NT“ sowie die Anzahl der Prozessoren des Rechners. Systemvariablen können sowohl über den Namen der Variable, als auch über eine Nummer ausgegeben werden. So liefert zum Beispiel die Eingabe „*Environ(17)*“ und *Environ(„NUMBER_OF_Processors“)* in meinem Fall jeweils den Wert „4“.

Das Direktfenster – Die Testhilfe

Dieses Fenster kann über das Menü „*Ansicht*“ und dem Befehl „*Direktfenster*“ oder über die Tastenkombination *[Strg] + [G]* geöffnet werden. Es befindet sich unter dem „*Codefenster*“. Verwendet wird dieses zum Testen von Programmen oder um direkt Befehle einzugeben, ohne ein eigenes Makro schreiben zu müssen. (vgl. Held, 2016, S. 12f.)

Klicken Sie mit der Maus in das „*Direktfenster“*. Geben Sie den Befehl „*?Environ("NUMBER_OF_Processors")“* ein und drücken Sie *[Enter]*. *Geben Sie den Befehl „?Environ(17)“* ein und drücken Sie *[Enter]*. Sie erhalten dabei nachstehenden Ausgaben im „*Direktfenster“*.

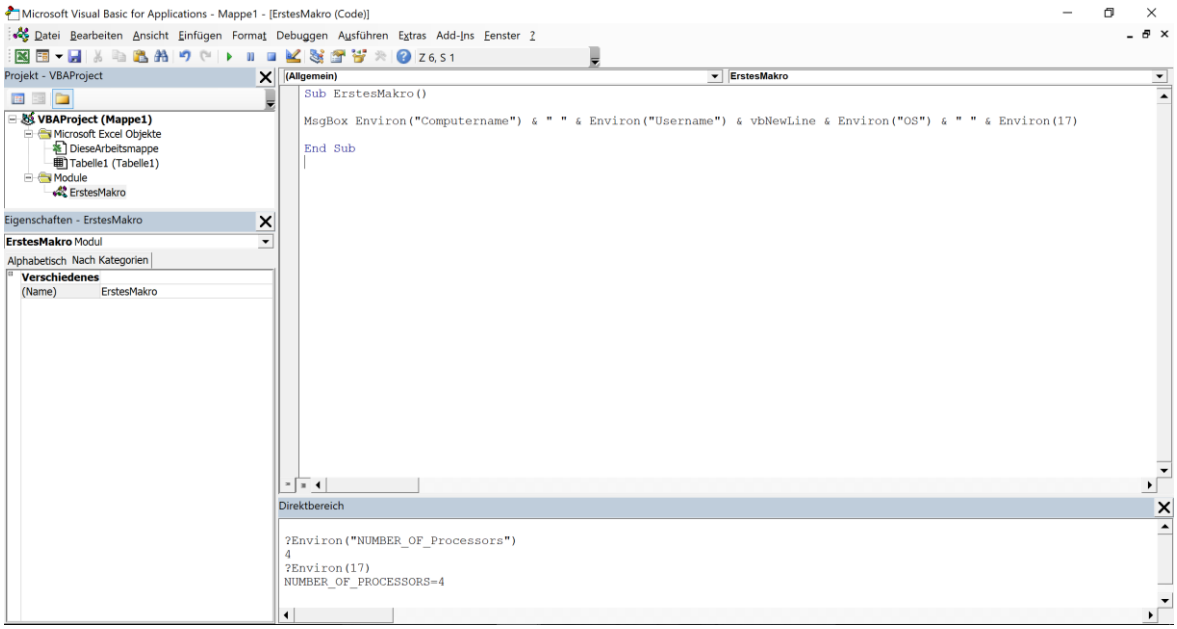


Abbildung 13: Das Direktfenster

Hinweis: Es können die möglichen Systemvariablen mit den Nummern eins bis 47 abgefragt werden!

Im „*Direktfenster*“ können aber auch diverse Eigenschaften von Tabellen oder Zellen geändert werden.

Durch die nachstehenden Eingaben und jeweiliges Drücken von [*Enter*] werden die Zellen „*A1*“, „*A2*“, „*A3*“ und „*A4*“ der „*Tabelle1*“ geändert, wobei die Schreibweise *Zelle*(*Zeile, Spalte*) zu beachten ist. Wird die Tabelle nicht angegeben, so wird in die aktive Tabelle geschrieben.

```
tabelle1.cells(1,1)="5"
tabelle1.cells(1,2)="3"
tabelle1.cells(1,3)=cells(1,1)*cells(1,2)
```

Die vorstehende Zeile errechnet das Produkt der Zellen „*A1*“ und „*B1*“ und gibt das Ergebnis in die Zelle „*C1*“ aus. Bei der nachstehenden Programmzeile hingegen wird die Formel und die Zelle D1 geschrieben.

```
tabelle1.cells(1,4)="=A1*B1"
```

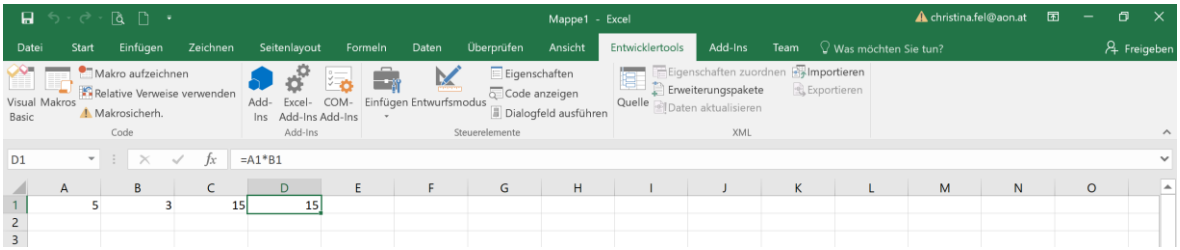


Abbildung 14: Ausgaben im Tabellenblatt nach eingaben im Direktfenster

Der Objektkatalog

Innerhalb der Entwicklungsumgebung befindet sich ein Nachschlagewerk, in welchem alle VBA-Befehle hinterlegt sind. Dieses wird durch das Menü „*Ansicht*“ und dem Befehl „*Objektkatalog*“ geöffnet. (vgl. Held, 2016, S. 16)

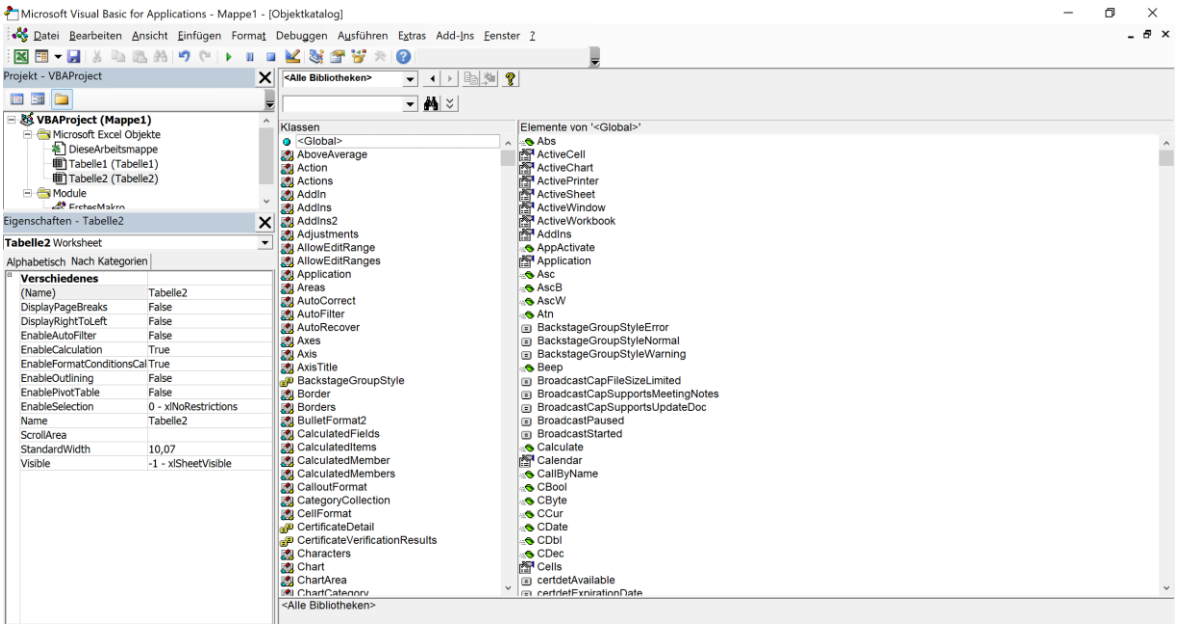


Abbildung 15: Objektkatalog als Auflistung aller verfügbaren Befehle in VBA

Dieser Objektkatalog wirkt anfänglich durch die Vielzahl an Einträgen sehr abschreckend. Man braucht aber im normalen Umgang nur einen Bruchteil der Befehle, um einen Großteil der Arbeiten in MS-Excel durchführen zu können. Wir betrachten folgend die kleinste Einheit in MS-Excel, das Objekt Zelle (=Range). Sie kommen zu den Elementen von „Range“ indem Sie in das Feld „*Klassen*“ klicken. Nach der Eingabe eines „r“ gelangen Sie automatisch zur Klasse „Range“. Sie können diese aber auch mit Hilfe der Bildlaufleiste suchen. Auf der rechten Seite sehen Sie nun alle Elemente der Klasse „*Range*“. (vgl. Held, 2016, S. 16f.)

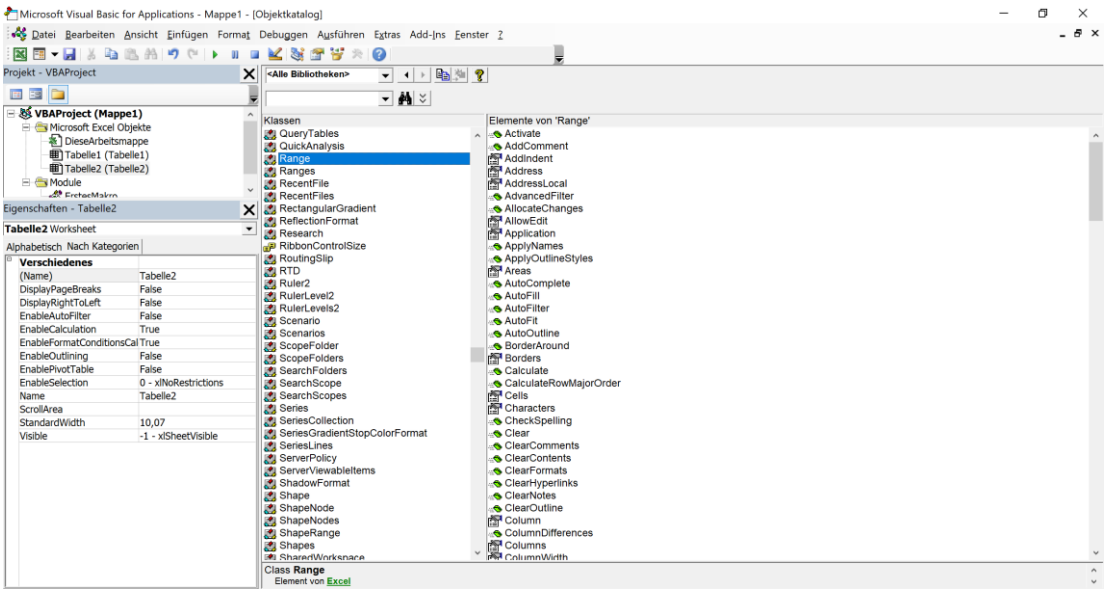


Abbildung 16: Die Elemente der Klasse "Range" aus dem Objektkatalog

Im über dem Feld „*Klassen*“ liegenden Drop-Down-Feld können Sie die Bibliotheken, in welchen nachgeschlagen wird, einschränken. Wechseln Sie zum Beispiel auf die Bibliothek „VBA“, so finden Sie dort in der Klasse „*Constants*“ das vorher verwendete Element „*vbNewLine*“. Es empfiehlt sich diesen Klassen etwas Zeit zu widmen und durchzusehen. Wählt man ein Element aus und drückt die Taste *[F1],* so gelangt man zu einer Onlinehilfe von Microsoft®.

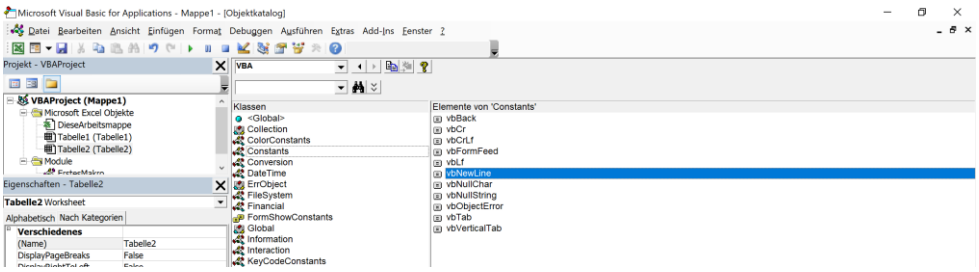


Abbildung 17: Die Elemente der Klasse "Constants" der Bibliothek "VBA" aus dem Objektkatalog

Die Operatoren

Operatoren dienen zum Zusammensetzen von Ausdrücken. Man unterscheidet dabei den „*Zuweisungsoperator*“ und den „*Verkettungsoperator*“ sowie die „*Arithmetischen Operatoren*“, „*Vergleichsoperatoren*“, und „*Logische Operatoren*“. Operatoren besitzen unterschiedliche Prioritäten, welche die Reihenfolge der Abarbeitung regeln. Diese Reihenfolge der Abarbeitung kann durch das gezielte Setzen von Klammern aber auch explizit gesetzt werden. (vgl. Theis , 2016, S. 112f.)

Zuweisungsoperator

Als Zuweisungsoperator fungiert das „=“-Zeichen. Dieses weist einer Variable einen Wert zu. „a=3“ bedeutet also, dass der Variable „*a*“ der Wert „*3*“ zugewiesen wird. Es ist aber zu beachten, dass „=“ auch ein Vergleichsoperator ist (siehe unten unter „Vergleichsoperatoren“).

Arithmetische Operatoren

Operator	Bezeichnung/ Beschreibung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
\	Ganzzahldivision
Mod	Modulo
^	Potenzierung

Abbildung 18: Arithmetische Operatoren

Die Rechenoperatoren „+, -, * und /“ liefern die altbekannten Ergebnisse. Sie müssen aber, um böse und unerwartete Ergebnisse in den Berechnungen zu vermeiden, auf die verwendeten Datentypen der Variablen achten. Sehen sie dazu bitte den Punkt "Verwendung von Variablen" dieses Handbuches an.

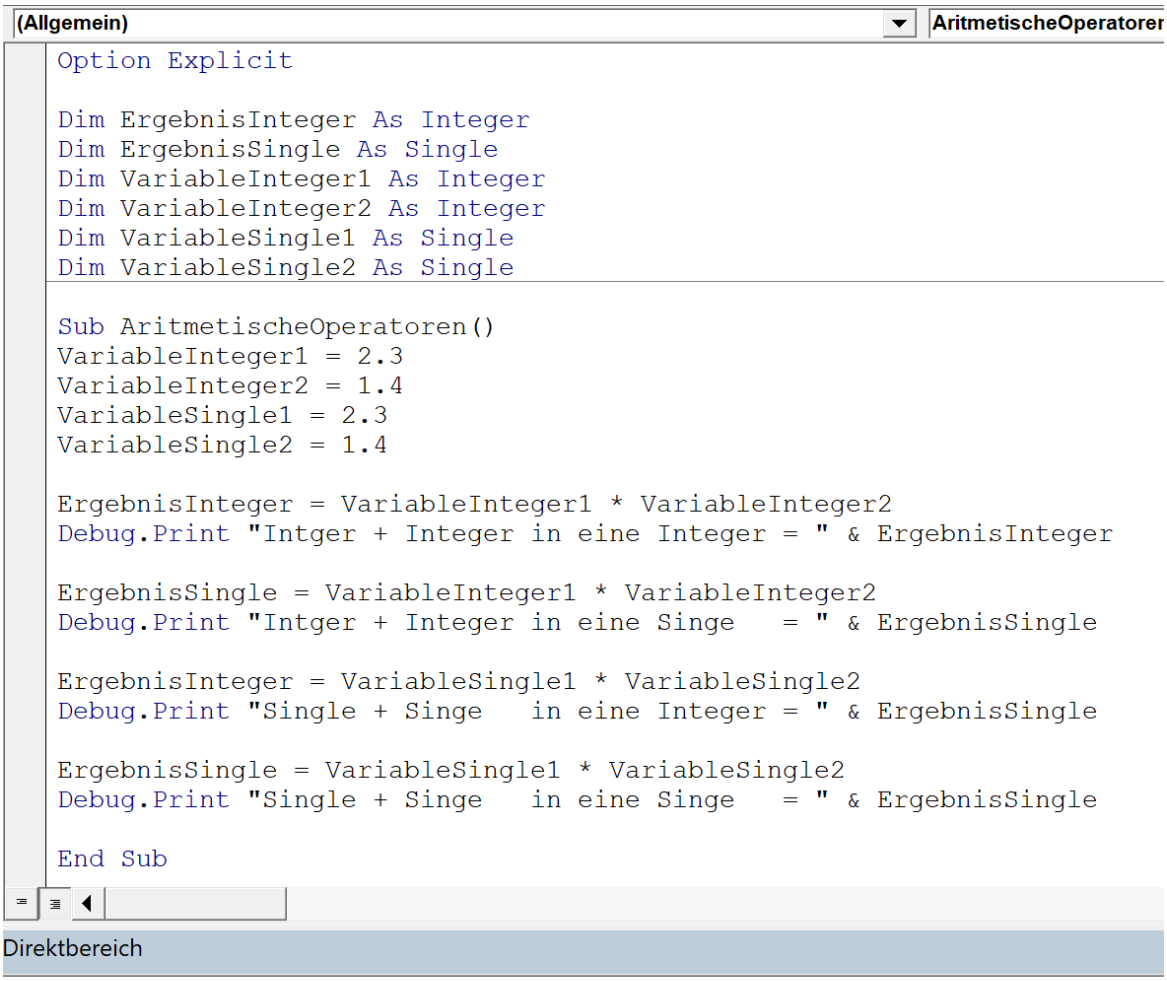


Abbildung 19: Programmcode und Ausgabe im Direktbereich, Zusammenspiel Arithmetische Operatoren und Datentypen von Variablen

Beachten Sie, dass in diesem Fall nur unter Verwendung der Variablen vom Type „Single“ ein rechnerisch korrektes Ergebnis geliefert wird. Es kann aber bei Programmen durchaus auch gewollt sein, dass Nachkommastellen unberücksichtigt bleiben.

Der Operator *„\“-Ganzzahldivision*“ arbeitet in zwei Schritten. Zuerst werden der „*Dividend“* und der „*Divisor*“ gerundet (Aufrundung ab 5). Danach wird das Ergebnis aus der Division ebenfalls gerundet. So ergibt die Rechnung „18 \ 4,7“, also nach der ersten Rundung „*18 / 5*“ den abermals gerundeten Wert „*3*“.

Der Operator „*Mod*“-Modulo geht ebenfalls in zwei Schritten vor. Zuerst werden der „*Dividend*“ und der „*Divisor*“ einzeln gerundet. Danach wird mit den gerundeten Werten eine Ganzzahlendivision durchgeführt und der verbleibende Rest ausgegeben.

Bei der Rechnung „*19,6 Mod 3,1*“ erfolgt also zuerst die Rundung auf die Werte 20 und 3. Der verbleibende Rest der Division „*20/3*“, also der Wert zwei, ist das Ergebnis der Berechnung.

Bei der „^“-Potenzrechnung können sowohl die „*Basis*“ als auch der „*Exponent*“ negativ und positiv sein. Es werden die jeweils richtigen Ergebnisse geliefert.

Vergleichsoperatoren

Vergleichsoperatoren werden zur Programmsteuerung benötigt. Diese steuern den Programmablauf in Verzweigungen und in Schleifen. Das Ergebnis der Bedingung ist entweder der Wert „Wahr“ oder „Falsch“ bzw. „True“ oder „False“, welche für Variablen des Typs „Boolean“ verwendet werden können. (vgl. Theis , 2016, S. 116f.)

Beim Operator „*Like*“ können Platzhalter verwendet werden, wobei „*“ für eine beliebige Anzahl an Zeichen steht. „?“ steht hingegen für genau ein Zeichen einer Zeichenkette. Dieser Operator dient dazu um Mustervergleiche von Zeichenketten durchführen zu können.

Operator	Bezeichnung/ Beschreibung
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
=	gleich
<>	ungleich
like	gleiche Zeichenfolge

Abbildung 20: Vergleichsoperatoren

```
(Allgemein)                                    Vergleisoperatoren
Option Explicit
Dim VariableInteger1 As Integer
Dim VariableInteger2 As Integer
Dim VariableText As String
Dim VareableBoolena1 As Boolean
Sub Vergleisoperatoren()
VariableInteger1 = 2
VariableInteger2 = 5
VariableText = "abcba"

VareableBoolena1 = VariableInteger1 > VariableInteger2
Debug.Print VareableBoolena1

VareableBoolena1 = VariableInteger1 < VariableInteger2
Debug.Print VareableBoolena1

VareableBoolena1 = VariableInteger1 + VariableInteger2 > 12
Debug.Print VareableBoolena1

VareableBoolena1 = VariableText Like "a*a"
Debug.Print VareableBoolena1

VareableBoolena1 = VariableText Like "a?a"
Debug.Print VareableBoolena1

VareableBoolena1 = VariableText Like "ab?a"
Debug.Print VareableBoolena1
End Sub
```

Abbildung 21: Vergleichsoperatoren Programmcode

```
Direktbereich
Falsch
Wahr
Falsch
Wahr
Falsch
Falsch
```

Abbildung 22: Vergleichsoperatoren Ausgabe Direktbereich

Logische Operatoren

Mit der Hilfe von „*Logischen Operatoren*“ können mehrere Bedingungen zusammengefasst werden.

Operator	Bezeichnung/ Beschreibung	Liefert Wahr (True) wenn
Not	Nicht	... der Ausdruck "Wahr" ist.
And	Und	... beide Ausfrucke "Wahr" sind.
Or	inklusives "oder"	... minestens ein Ausdruck "Wahr" ist.
Xor	Exclusives "oder"	... genau ein Ausdruck "Wahr" ist.

Abbildung 23: Auswahl der wichtigsten logischen Operatoren

Die Funktionsweise der logischen Operatoren kann aus nachstehender Wahrheitstabelle ersehen werden.

Ausdruck 1	Ausdruck 2	Ergebnis Ausdruck 1	Ergebnis Ausdruck 2	Not (bezogen auf Ausdruck 1)	And	Or	Xor
2>1	3<4	WAHR	WAHR	FALSCH	WAHR	WAHR	FALSCH
2>1	4<3	WAHR	FALSCH	FALSCH	FALSCH	WAHR	WAHR
1>2	3<4	FALSCH	WAHR	WAHR	FALSCH	WAHR	WAHR
1>2	4<3	FALSCH	FALSCH	WAHR	FALSCH	FALSCH	FALSCH

Abbildung 24: Wahrheitstabelle logische Operatoren

Abbildung 25: Logische Operatoren Programmcode und Ausgabe im Direktbereich

Der vorstehende Programmcode veranschaulicht die Arbeitsweise der „*Logischen Operatoren*“. Anhand der Wahrheitstabelle (siehe Abbildung 24: Wahrheitstabelle logische Operatoren) können die vorstehenden ausgegebenen Ergebnisse verglichen werden.

Verkettungsoperator

Mit Hilfe des Verkettungsoperators „&“ können Zeichenketten miteinander verbunden werden. Er dient zur ansprechenden Darstellung allfälliger Ergebnisse aus Berechnungen. Wenn einer der verwendeten Ausdrucke keine Zeichenfolge ist, so wird dieser automatisch in eine Zeichenkette geändert. Die Gesamtausgabe stellt eine zusammengefasste Zeichenkette dar. Verwendete Variablen bleiben aber in ihrer ursprünglichen Form erhalten. (vgl. Theis , 2016, S. 120)

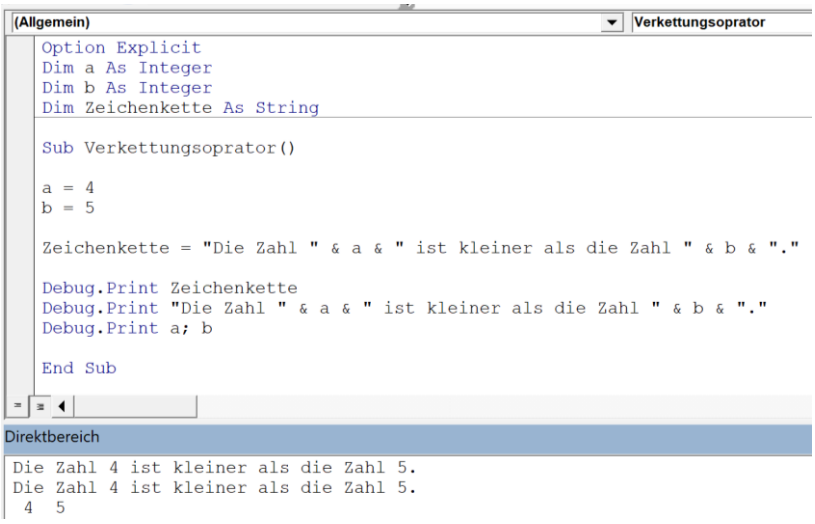


Abbildung 26: Verkettungsoperator Programmcode und Ausgabe im Direktbereich

Aus dem vorstehenden Programmcode ist die Verkettung von Zeichenketten und Variablen zu einer neuen Zeichenkette ersichtlich. Es kann diese verkettete Zeichenkette einer eigenen Variable (Typ „*String*“) zugewiesen werden. Die erste Zeile im Direktbereich stellt die Ausgabe des Textes aus der Zuweisung der verknüpften Zeichenkette in die Variable „Zeichenkette“ dar, wogegen die zweite Zeile im Direktbereich ausgegeben wird, ohne diese Zeichenkette einer Variable zuzuweisen. Dass alle verwendeten Ausgangsvariablen unverändert bleiben, kann aus der dritten Zeile im Ausgabebereich ersehen werden.

Rangfolge der Operatoren

Wenn ein Ausdruck aus mehreren Operatoren besteht, so werden die einzelnen Teilausdrücke entsprechend der „Priorität der Operatoren“ abgearbeitet. Operatoren innerhalb von Klammern haben die größte Priorität und werden zuerst abgearbeitet. Innerhalb der Klammern kommt wieder die normale Regelung der Priorität zur Anwendung. Ausdrucke mit Operatoren mit gleicher Priorität werden von links nach rechts abgearbeitet.

Operator	Bezeichnung/ Beschreibung
()	Eingeklammerter Ausdruck
^	Potenz
-	negatives Vorzeichen
* und /	Multiplikation und Division
\	Ganzzahldivision
Mod	Modulo
+ und -	Addition und Subtraktion
&	Verkettung
=, <, >, <>, <=, >= und Like	Vergleichsoperatoren ("=" als Vergleich und nicht als Zuweisung)

Abbildung 27: Priorität der Operatoren

Befehle (Anweisungen und Fuktionen)

Im Folgenden finden Sie eine Auflistung über die gängigsten VBA-Anweisungen und Funktionen mit einer Kurzbeschreibung, welche in wenigen Worten aussagt, wofür diese verwendet werden können. Weitere Informationen zu den Anweisungen und Funktionen (Syntax, Parameter) finden Sie im Hilfesystem von Excel. (vgl. Walkenbach, 2016, S. 1f.)

Befehl	Type	Kurzbeschreibung
Abs	Funktion	Rückgabe des Betrages einer Zahl.
AppActivate	Anweisung	Ein Anwendungsfenster wird aktiviert.
Array	Funktion	Wert vom Typ Variant eines Feldes.
Asc	Funktion	Rückgabe des ASCII-Wertes des ersten Buchstabens eines Strings.
Atn	Funktion	Rückgabe des Arkustangens einer Zahl.
Beep	Anweisung	Ein Ton wird wüber den Computerlautsprecher ausgegeben.
Call	Anweisung	Steuerung wird auf eine andere Prozedur übertragen.
CBool	Funktion	Umwandlung eines Ausdruckes in den Typ in Boolean.
CByte	Funktion	Umwandlung eines Ausdruckes in den Typ in Byte.
CCur	Funktion	Umwandlung eines Ausdruckes in den Typ Currancy.
CDate	Funktion	Umwandlung eines Ausdruckes in den Typ Date.
CDbl	Funktion	Umwandlung eines Ausdruckes in den Typ Double.
CDec	Funktion	Umwandlung eines Ausdruckes in einen Dezimalwert.
ChDir	Anweisung	Wechsel auf das aktuelle Verzeichnis.
ChDrive	Anweisung	Wechsel auf das aktuelle Laufwerk.
Choose	Funktion	Auswahl eines Wertes aus einer Argumentenliste.
Chr	Funktion	Umwandlung eines ANSI-Wetes in ein String.
CInt	Funktion	Umwandlung eines Ausdruckes in den Typ Integer.
CLng	Funktion	Umwandlung eines Ausdruckes in den Typ Long.
Close	Anweisung	Eine Textdatei wird geschlossen.
Const	Anweisung	Dekleration einer Konstante.
Cos	Funktion	Rückgabe des Kosinus einer Zahl.
CreateObject	Funktion	Erzeugung eines OLE-Automationsobjektes.
CSng	Funktion	Umwandlung eines Ausdruckes in den Typ Single.
CStr	Funktion	Umwandlung eines Ausdruckes in den Typ String.
CurDir	Funktion	Rückgabe des aktuellen Pfades.
CVar	Funktion	Umwandlung eines Ausdruckes in den Typ Variant.
CVDate	Funktion	Umwandlung eines Ausdruckes in den Typ Date.
CVErr	Funktion	Rückgabe eines benutzerdefinierten Fehlercodes.
Date	Anweisung	Einstellung des aktuellen Systemdatums.
Date	Funktion	Rückgabe des aktuellen Systemdatums.
DateAdd	Funktion	Rückgabe eines Datums, zu welchem ein bestimmtes zeitliches Intervall addiert wurde.
DateDiff	Funktion	Rückgabe eines Datums, von welchem ein bestimmtes zeitliches Intervall subtrahiert wurde.
DatePart	Funktion	Rückgabe eines Integer-Wertes, welcher ein bestimmtes Datum enthält.
DateSerial	Funktion	Rückgabe eines seriellen Wertes eines Datums
Datevalue	Funktion	Umwandlung einer Zeichenfolge in ein Datum.
Day	Funktion	Rückgabe des Tages eines Datums

Abbildung 28: Wichtige VBA-Anweisungen Teil 1 (vgl. Walkenbach, 2016, S. 1ff.)

Befehl	Type	Kurzbeschreibung
Declare	Anweisung	Deklarationen einer Referenz auf eine externe Prozedur (DLL).
DeleteSetting	Anweisung	Löschung der Einträge einer Anwendung aus der Windows Registrierung.
Dim	Anweisung	Deklaration von Variablen, optional auch des Datentyps
Dir	Funktion	Rückgabe des Namen einer Datei oder eines Verzeichnisses, welche einem bestimmten Muster entsprechen.
Do- Loop	Anweisung	Anweisungsblock wird durchlaufen (Ausstieg mit Exit).
DoEvents	Funktion	Kontrolle wird an das Btriebssystem übergeben, um andere Ereignisse abzuarbeiten.
End	Anweisung	Beendet das Programm oder einen Anwesungsblock (If, With, Sub, Function, Property, Type oder Select)
EOF	Funktion	Rückgabe von True bei Erreichung des Endes einer Textdatei.
Erase	Anweisung	Neuinitialisierung eines Arrys.
Error	Anweisung	Simulation einer spezifischen Fehlerbedingung.
Error	Funktion	Rückgabe eines Fehlermeldung, weche einem bestimmten Fehlercode zugeordnet ist.
Exit Do	Anweisung	Do-Loop-Coeblock wird verlassen.
Exit For	Anweisung	For-Next-Codeblock wid verlassen.
Exit Function	Anweisung	Eine Funktionsprozedur wird verlassen.
Exit Property	Anweisung	Eine Eigenschaftsprozedur wird verlassen.
Exit Sub	Anweisung	Eine Subprozedur wird verlassen.
Exp	Funktion	Rückgabe eines Wertes, welcher die Zahl "e" potenziert mit einem Wert
FileAttr	Funktion	Rückgabe eines Wertes, welcher den Zugriffsmodus einer geöffneten Datei darstellt.
FileCopy	Anweisung	Datei wird kopiert.
FileDateTime	Funktion	Rückgabe des Zeitpunktes, an welchem eine Datei zuletzt geändert wurde.
FileLen	Funktion	Rückgabe der Länge einer Datei.
Fix	Funktion	Rückgabe des Integer-Anteiles einer Zahl.
For Each -Next	Anweisung	Anweisungsblock wird für jedes Element eines Objektes ausgeführt.
For -Next	Anweisung	Anwesungsblock wird eine bestimmte Anzahl oft durcgeführt.
Format	Funktion	Anzeige eines Ausdruckes in einem gewählten Format.
FormatCurrency	Funktion	Rückgabe einer Zahl als String, welche als Währung formatiert ist.
FormatDateTime	Funktion	Rückgabe einer Zahl als String, welche als Datum formatiert ist.
FormatNumber	Funktion	Rückgabe einer Zahl als String formatiert.
FormatPercent	Funktion	Rückgabe einer Zahl als String, welche als Prozentangabe formatiert ist.
FreeFile	Funktion	Rückgabe der nächsten Dateinummer bei Verwendung des Befehles Open.
Function	Anweisung	Deklaration von Namen und Argumenten einer Funktionsprozedur.
Get	Anweisung	Daten aus einer Textdatei werden gelesen.
Get0bject	Funktion	Rückgabe eines OLE-Automationsobjektes aus einer Datei.
GetAll	Funktion	Rückgabe der Schlüsseleinstellungen einer Anwendung in der Windows-Registrierung.
GetAttr	Funktion	Rückgabe des Codes der die Dateiattrubute abbildet.
GetSetting	Funktion	Rückgabe einer Schlüsseleinstellung aus der Windows-Registry.
.... GoSub	Anweisung	Verzweigung wird aufgerufen.
GoSub. . .Return	Anweisung	Verzweigung in eine andere Prozedur mit anschließener Rückkehr.
GoTo	Anweisung	Verzweigung innerhalb einer Prozedur an eine angegebene Stelle.
Hex	Funktion	Umwandlung einer dezimaen Zahl in hexadezimal.
Hour	Funktion	Rückgabe der Stunde einer Urzeitangabe.
If-Then- Else	Anweisung	Bedingte Verarbeitung von Anweisungen.
IIF	Funktion	Rückgabe eines von zwei Werten als Ergebnis eines Ausdruckes.
Input	Funktion	Rückgabe einer angebenenen Anzahl von Zeichen einer Textdatei.

Abbildung 29: Wichtige VBA-Anweisungen Teil 2 (vgl. Walkenbach, 2016, S. 1ff.)

Befehl	Type	Kurzbeschreibung
InputBox	Funktion	Anzeige eines Eingabedialoges.
InStr	Funktion	Rückgabe der Position einer Zeichenfolge von einer anderen Zeichenfolge.
InStrRev	Funktion	Rückgabe der Position einer Zeichenfolge von einer anderen Zeichenfolge vom Ende aus gesehen..
Int	Funktion	Rückgabe des Integer-Anteiles einer Zahl.
Is0bject	Funktion	Rückgabe von True, wenn die Variable ein Objekt ist.
IsArray	Funktion	Rückgabe von True, wenn die Variable ein Feld ist.
IsDate	Funktion	Rückgabe von True, wenn die Variable ein Datum ist.
IsEmpty	Funktion	Rückgabe von True, wenn die Variable initialisiert ist.
IsError	Funktion	Rückgabe von True, wenn der Ausruck ident mit einem Fehlerausdruck ist.
IsMissing	Funktion	Rückgabe von True, wenn ein optionales Argument fehlt.
IsNull	Funktion	Rückgabe von True, wenn der Ausdruck keine Daten enthält.
IsNumeric	Funktion	Rückgabe von True, wenn die Variable eine Zahl ist.
Join	Funktion	Rückgabe eines String, welcher durch Zusammenfügung von Strings eines Arrays entstand.
Kill	Anweisung	Eine Datei wird gelösscht.
LBound	Funktion	Rückgabe der unteren Grenze eines Feldes.
LCase	Funktion	Rückgabe eines Strings in Kleinbuchstaben.
Left	Funktion	Rückgabe einer bestimmten Menge von Zeichen eines String ganz links beginnend.
Len	Funktion	Rückgabe der Länge eines Strings in Zeichen.
Line Input #	Anweisung	Eine Zeile einer Textdatei wird eingelesen.
Load	Anweisung	Ein Objekt wird geladen, aber nicht angezeigt.
Loc	Funktion	Rückgabe der derzeitigen Position in einer Datei.
Lock . . . Unlock	Anweisung	Der Zugriff auf eine Textdatei wird gesteuert.
LOF	Funktion	Rückgabe der Zahl der Bytes in einer Textdatei.
Log	Funktion	Rückgabe des natürlichen Logarithmus einer Zahl.
LTrim	Funktion	Rückgabe éiner Zeichenkette ohne vorangehende Leerzeichen.
Mid	Anweisung	Zeichen einer Zeichenkette werden durch andere Zeichen ersetzt.
Mid	Funktion	Rückgabe einer bestemmten Menge von Zeichen eins Strings.
MidB	Funktion	Rückgabe einer bestemmten Menge von Bytes eins Strings.
Minute	Funktion	Rückgabe der Minuten einer Zeitangabe.
MkDir	Anweisung	Ein Verzeichnis wird erstellt.
Month	Funktion	Rückgabe des Monats einer Datumsangabe.
MonthName	Funktion	Rückgabe des Namen eines Monates.
MsgBox	Funktion	Anzeige eines Meldungsfensters.
Name	Anweisung	Eine Datei oder ein Verzeichnis wird benannt.
Now	Funktion	Rückgabe der aktuellen Sytemzeit.
Oct	Funktion	Umwandlung einer dezimaen Zahl in oktal.
On . . . GoTo	Anweisung	Verzweigung wird abhängig von einer Bedingung ausgeführt.
On Error	Anweisung	Im Fehlerfalle auszuführende Anweisungen.
Open	Anweisung	Eine Textdatei wird geöffnet.
Option Base	Anweisung	Die Untergrenze eine Arrays wird geändert
Option Compare	Anweisung	Zeichnketten werden im Standardvergleichsmodus verglichen.
Option Explicit	Anweisung	Ein gesaamtes Modul wird als Private geführt.
Print #	Anweisung	Daten werden in eine sequentielle Datei geschrieben.
Private	Anweisung	Deklaration einer loklen Variable oder einer lokalen Arrays.
Property Get	Anweisung	Deklaration des Namens und der Argumente einer Property Get-Prozedur.
Property Let	Anweisung	Deklaration des Namens und der Argumente einer Property Let-Prozedur.
Property Set	Anweisung	Deklaration des Namens und der Argumente einer Property Set-Prozedur.
Public	Anweisung	Deklaration einer öffentlichen Variable oder eines öffentlichen Arrays.

Abbildung 30: Wichtige VBA-Anweisungen Teil 3 (vgl. Walkenbach, 2016, S. 1ff.)

Befehl	Type	Kurzbeschreibung
Put	Anweisung	Variable wird in eine Textdatei geschrieben.
RaiseEvent	Anweisung	Auslösung eines benutzerdefinierten Ereignisses.
Randomize	Anweisung	Initialisierung des Zahlenzufallsgenerators.
ReDim	Anweisung	Änderung der Dimmensionen eines Arrays.
Rem	Anweisung	Definiert eine Kommentarzeile.
Replace	Funktion	Rückgabe eines Strings, bei welchem ein Teilstring durch einenen anderen String ersetzt wurde.
Reset	Anweisung	Schließung aller offenen Textdatein.
Resume	Anweisung	Fortsetzung einer Prozedur nachdem eine Fehlerbehandlungsroutine abgschlossen wurde.
RGB	Funktion	Rückgabe des Wertrs einer RGB-Fahrbe.
Right	Funktion	Rückgabe einer bestimmten Menge von Zeichen eines String ganz rechts beginnend.
RmDir	Anweisung	Löschung eines leeren Verzeichnisses.
Rnd	Funktion	Rückgbe einer Zufallszahl zwischen 0 und 1
Round	Funktion	Rückgabe einer auf die angegebene Anzahl von Dezimalstellen gerundeten
RTrim	Funktion	Rückgabe einer Zeichenkette ohne nachfolgende Leerzeichen.
SaveSetting	Anweisung	Speicherung oder Erstellung eines Eintrages in die Windows-Registrierung.
Second	Funktion	Rückgabe der Sekunden einer Uhrzeitangabe.
Seek	Funktion	Rückgabe der Position in einer Textdatei.
Seek	Anweisung	Festlegung der Position für den nächsten Zugriff in einer Textdatei.
Select Case	Anweisung	Bedinge Verarbeitung von Anweisungen.
SendKeys	Anweisung	Tastendrucke werden an das aktive Fenster gesendet.
Set	Anweisung	Zuweisung einer Objektreferenz an eine Variable oder Eigenschaft.
SetAtt r	Anweisung	Änderung der Dateiatrribute.
Sgn	Funktion	Rückgabe eines Integerwertes, der das Vorzeichen einer Zahl beschreibt.
Shell	Funktion	Aufruf eines Programmes.
Sin	Funktion	Rückgabe des Sinus einer Zahl.
Space	Funktion	Rückgabe einer Zeichenfolge mit einer bstimmten Anzahl von Leerzeichen.
Split	Funktion	Rückgabe eines Feldes welches aus Teilstrings besteht.
Sqr	Funktion	Rückgabe der Quadratwurzel einer Zahl.
Static	Anweisung	Dekleration von Variablen auf Prozedurebene, Werte bleiben auf die Dauer der Laufzeit der Prozedur erhalten.
Stop	Anweisung	Programm wird unterbrochen
Str	Funktion	Rückgabe der Repräsentation einer Zahl als String.
StrComp	Funktion	Rückgabe eines Werts als Ergebnis des Vergleiches von zwei Zeichenfolgen.
StrConv	Funktion	Rückgabe eines umgewandelten Strings.
String	Funktion	Rückgabe sich wiederholender Zeichen oder Zeichenfolgen.
StrReverse	Funktion	Rückgabe einer Zeichenfolge in umgekehrter Reiehenfolge.
Sub	Anweisung	Deklaration des Namens und der Argumente einer Sub-Prozedur.
Switch	Funktion	Auswertung einer Liste von Ausdrücken.
Tab	Funktion	Positionierung der Ausgabe.
Tan	Funktion	Rückgabe des Tangens eines Wertes.
Time	Anweisung	Eintellung der Systemzeit.
Time	Funktion	Rückgabe der aktuellen Systemzeit.
Timer	Funktion	Rückgabe der Anzahl der vergangenen Sekunden seit Mitternacht.
Timeserial	Funktion	Rückgabe der Uhrzeit der angegebenen Stunden, Minuten sowie Sekunden.
Timevalue	Funktion	Umwandlung eines Strings in Time.
Trim	Funktion	Rückgabe eines String ohne vor- und nachgestellte Leerzeichen.
Type	Anweisung	Definition eines benutzerdefinierten Datentyps.
TypeName	Funktion	Rückgabe des Types einer Variable als String.

Abbildung 31: Wichtige VBA-Anweisungen Teil 4 (vgl. Walkenbach, 2016, S. 1ff.)

Befehl	Type	Kurzbeschreibung
UBound	Funktion	Rückgabe der obere Grenze eines Feldes.
UCase	Funktion	Rückgabe eines Strings in Großbuchstaben.
Unload	Anweisung	Entfernung eines Objektes aus dem Speicher.
Val	Funktion	Rückgabe der Zahlen einer Zeichenkette.
VarType	Funktion	Rückgabe einer Zahl die dem Typ einer Variable angibt..
weekday	Funktion	Rückgabe einer Zahl welchen einen Wochentag symbolisiert.
WeekdayName	Funktion	Rückgabe des Namens eines Wochentages.
while . . . wend	Anweisung	Durchlauf eines Anweisungsblockes, solange eine bestimmte Bedingung erfüllt ist.
width #	Anweisung	Breite einer Ausgabezeile einer Textdatei wird eingestellt.
with	Anweisung	Zugriff auf mehrere Eigenschaften eines Objektes.
write #	Anweisung	Daten werden in eine sequentielle Textdatei geschrieben.
Year	Funktion	Rückgabe des Jahres eines Datums.

Abbildung 32: Wichtige VBA-Anweisungen Teil 5 (vgl. Walkenbach, 2016, S. 1ff.)

Erste Interaktionen – Eingabe und Ausgabe von Daten

Die Eingabe und Ausgabe von Daten kann einerseits über eine Excel Arbeitsmappe und andererseits über Eingabeboxen erfolgen.

In einem ersten Schritt werden wir den Zellen „*A1*“ den Wert 100 bzw. „*A2*“ den Wert 30 zuweisen. In den Zellen „*A3*“ soll das Produkt und in „*A4*“ die Summe der Zellen „*A1*“ und „*A2*“ ausgegeben werden.

Wählen Sie im Menü „*Einfügen*“ den Befehl „*Modul*“ aus. Ändern Sie im Eigenschaftsfenster den Namen auf „*ErsteEingaben*“. Fügen Sie nachstehenden Code in das, „Codefenster“ein:

```
Sub ErsteEingaben()
Cells(1, 1) = 100
Range("B1").Value = 30
Cells(1, 3) = Cells(1, 1) * Cells(1, 2)
Cells(1, 4) = Cells(1, 1) + Range("B1")
End Sub
```

Sie sehen, dass auf einzelne Zellen, sowohl über das *Objekt* „*Range*“, als auch die *Methode* (auch Eigenschaft) „*Cells*“ zugegriffen werden kann.

Durch das Drücken der Taste *[F5]* oder der Schaltfläche  wird das Makro gestartet.

Wechseln Sie nun zum Excel-Arbeitsblatt und Sie erkennen, dass das Makro korrekt ausgeführt wurde.

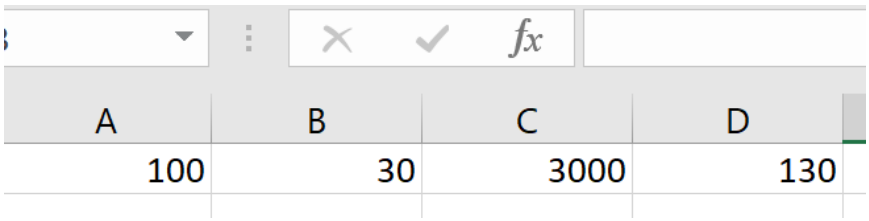


Abbildung 33: Arbeitsmappe nach Ablauf des Makros ErsteEingaben

Um den Komfort und die Bedienerfreundlichkeit zu erhöhen, werden wir nunmehr einen Button *„Start“* hinzufügen und mit dem Makro „*ErsteEingaben*“ verknüpfen. Drücken Sie dazu im Menü „*Entwicklertools*“ auf die Schaltfläche 

und wählen Sie mit der

Schaltfläche XXXX eine *Schaltfläche (Steuerelement)* aus. Markieren Sie die Position und die gewünschte Größe der Schaltfläche im Bereich der Excel-Arbeitsmappe. Nach dem Loslassen der linken Maustaste erscheint ein Fenster mit einer Auswahlliste, wählen Sie das vorher erstellte Makro „*ErsteEingaben*“ und drücken Sie die Schaltfläche *[OK]*. Durch Klicken der Schaltfläche mit der rechten Maustaste können Sie im Entwurfsmodus die Aufschrift auf „*Makro Start*“ ändern. Leeren Sie den Bereich „A1“ bis „A4“ und betätigen Sie den vorher erstellten Button. Das Makro „*ErsteEingaben*“ wird ausgeführt.

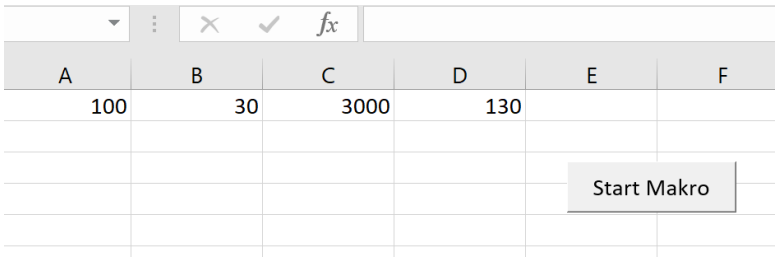


Abbildung 34: Excel Arbeitsmappe nach Betätigung des Buttons "Start Makro"

Die Ausgabe kann aber auch auf eine „*MessageBox*“ geleitet werden.

Ergänzen Sie im Makro „*ErsteEingaben*“ nachstehende Befehlszeile:

```
MsgBox "Zelle A1 - Zelle A3 = " & Cells(1, 1) - Cells(1, 2)
```

Starten Sie das Makro abermals durch Klicken der Schaltfläche

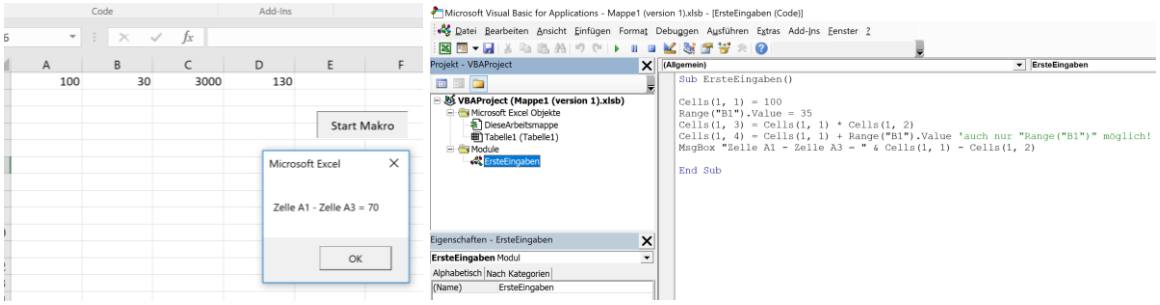


Abbildung 35: Excel Arbeitsmappe mit MessageBox nach Betätigung des Buttons "Start Makro"

Methoden, Eigenschaften und Ereignisse eines Objektes

Da VBA eine objektorienterte Progrmmiersprache ist, wird mit Objekten gearbeitet, welche über „*Eigenschaften*“, „*Methoden*“ und „*Ereignisse*“ verfügen. (vgl. Theis , 2016, S. 48)

Die *Eigenschaften* (Attibute) eines Objektes bestimmen das Aussehen. So besitzt z.B. ein Tabellenblatt die *Eigenschaft* „*Name*“, dessen Wert die Bezeichnung des Tabellenblattes (z.B. „Tabelle1“) darstellt. (vgl. Theis , 2016, S. 48)

Die Abfrage z.B. erfolgt im Direktfenster über die Eingabe

```
?Worksheets("Tabelle1").Name
```

Die *Methoden* eines Objektes legen dessen Fähigkeiten fest. So besitzt zum Beispiel eine Zelle die *Methode* „*copy*“, was ermöglicht, dass der Inhalt kopiert werden kann. (vgl. Theis , 2016, S. 48)

Die *Ereignisse* eines Objektes bestimmen, was mit diesem passiert. Ein Tabellenblatt besitzt z.B. das Ereignis der „*Aktivierung*“. Dieses Ereignis kann mit VBA-Code verknüpft werden, welcher beim Eintritt des Ereignisses ausgeführt wird. (vgl. Theis , 2016, S. 48)

Objekthierarchie in VBA

In VBA verfügen Objekte über Eigenschaften, welche wiederum Unterobjekte mit eigenen Eigenschaften sein können. Diese Logik weitergeführt führt zu einer Hierarchie von Objekten, welche von einem Hauptobjekt ausgehen. (vgl. Theis , 2016, S. 49)

Das Hauptobjekt stellt die „*Application*“ (Anwendung) *Excel* dar. Eine Eigenschaft des Objektes „*Application*“ stellt die Auflistung „*Workbooks*“ dar, in welcher sich alle geöffneten Arbeitsmappen (somit Excel Dateien) befinden. Eine Eigenschaft des Objektes „*Workbooks*“ stellt die Auflistung „*Worksheets*“ dar, in welcher sich alle Tabellenblätter einer Arbeitsmappe befinden. Eine Eigenschaft des Objektes „*Worksheet*“ ist das Objekt „*Range*“, in welchem sich Zellen oder Zellenbreiche eines Tabellenblattes befinden. (vgl. Theis , 2016, S. 50)

Auflistungen sind Sammlungen gleichartiger Objekte, wie zum Beispiel „*Workbooks*“. Diese Auflistungen sind leicht durch das „s“ am Ende erkennbar. (vgl. Theis , 2016, S. 50)

Verwendung von Variablen

Zur Speicherung vorübergehender Daten werden Variablen verwendet. Den Variablen werden vordefinierte Datentypen zugewiesen, welche jeweils besondere Eigenschaften haben. (Theis , 2016, S. 104)

In VBA kann man zwar prinzipiell ohne die vorherige Deklaration von Variablen arbeiten. Es empfiehlt sich aber auch bei VBA diese durchzuführen. Diese begründet sich einerseits mit einer Verbesserung der Performance der Programme samt einer Verringerung des Speicherbedarfes und andererseits mit der Tatsache, dass andere Programmiersprachen eine Dekleration zwingend erwarten und man es gleich richtig für einen allfälligen Umstieg lernt. Weiters können dadurch häufig Fehler vermieden werden. (Theis , 2016, S. 104)

Durch die Angabe der „*Option Explizit*“ im Deklerationsbereich eines Modules wird der Programmierer dazu gezwungen die Variablen zu deklarieren. Im VBA-Menü unter „*Extras*“ und „*Optionen*“ sollte deshalb unbedingt das Häkchen bei „*Variablendekleration erforderlich*“ gesetzt werden. (vgl. Das VBA-Tutorial, 2018) (vgl. Theis , 2016, S. 105)

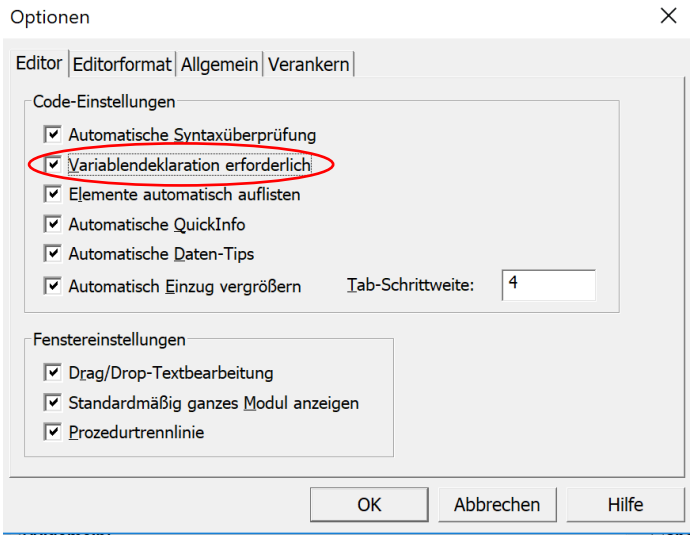


Abbildung 36: Optionen im VBA-Menü Extras

Deklarationen sind Informationsanweisungen, welche der Programmierer VBA zur Verfügung stellt. Es sind Anweisungen, welche für eine allgemeine Ordnung Sorge tragen und nicht abgearbeitet werden. (vgl. Walkenbach, 2016, S. 64)

Nachstehende Grafik zeigt die wichtigsten von VBA unterstützten Datentypen. (vgl. Das VBA-Tutorial, 2018)

Typ	Wertebereich	Speicherbedarf	Anfangswert	Beispiel
Boolean	Ja/Nein-Werte (True oder False)	2 Byte	False	BuchungBestätigt = True
Byte	Ganzzahlen 0 bis 255	1 Byte	0	Sitzreihe = 42
Integer	Ganzzahlen -32.768 bis 32.767	2 Byte	0	Sitzplätze = 480
Long	Ganzzahlen -2.147.483.648 bis 2.147.483.647	4 Byte	0	GesammelteMeilen = 46377
LongLong *	Ganzzahlen ca. ± 9 Trillionen	8 Byte	0	Flugnummer = 7577415118
Currency	skalierte Ganzzahlen ca. ± 9 Billionen, 4 Nachkommastellen	8 Byte	0	Preis = 815,00
Single	Gleitkommazahl, einfache Genauigkeit	4 Byte	0	Ergebnisgerundet = 2.7
Double	Gleitkommazahl, doppelte Genauigkeit	8 Byte	0	Ergebnis = 2.75
Date	Datum (1. Jan 100 bis 31. Dez 9999) und Zeit	8 Byte	30.12.1899 00:00	Buchungsdatum = #2/16/2018#
String	Texte	Anz. Zeichen * 2	""	Info = "Guten Flug"
Object	abhängig vom Objekt	4 Byte	Nothing	Set Emirates = Fluglinie
Variant	abhängig vom gerade aktuellen Inhalt	unterschiedlich	Empty	sonstiges = zB Verspätung 25 oder gestrichen
* nur auf 64 Bit Rechnern				

Abbildung 37: Die wichtigsten von VBA unterstützte Datentypen

Bei der Verwendung von Variablen muss beachtet werden, dass das erste Zeichen ein Buchstabe ist, keine Leerzeichen und bestimmte Sonderzeichen wie z.B. #, %, ?, ! oder & verwendet werden und es innerhalb des Gültigkeitsbereiches keine zwei Variablen mit demselben Namen geben kann. (vgl. Theis , 2016, S. 104)

Um Inhalte von Variablen zwischen zwei Makros übernehmen zu können ist es erforderlich, dass diese als „*Public*“ deklariert werden. Als „*Privat*“ deklarierte Variablen können nur innerhalb des Modules verwendet werden. Beim Verlassen des Makros werden als „*Privat*“ deklarierte Variablen gelöscht und deren Inhalte gehen verloren. (vgl. Held, 2016, S. 44)

Weiters dürfen Makros, welche Werte von Variablen übernehmen sollen, nicht als „*Private Sub*“ angelegt werden. Makros, welche Werte aus Variablen anderer Makros zur Verfügung stellen sollen müssen als „*Public Sub*“ angelegt werden.

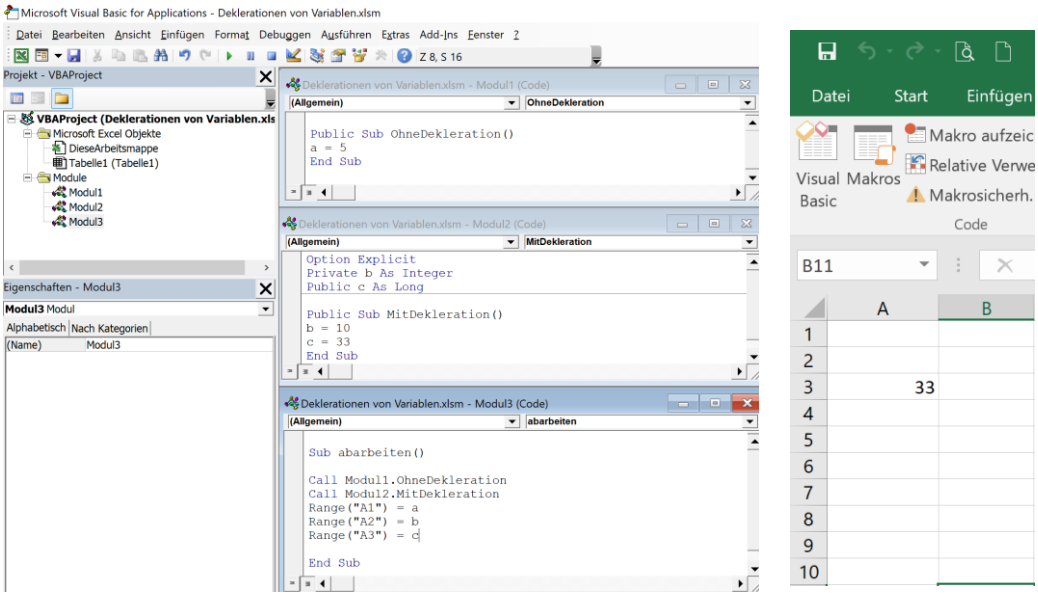


Abbildung 38: Unterschiedliche Deklarationen von Variablen und deren Auswirkungen

In den Makros „*OhneDekleration*“ und „*MitDekleration*“ werden den Variablen *a*, *b* und *c* Werte zugewiesen. Im Makro „*abarbeiten*“ werden die vorgenannten Makros zuerst aufgerufen. Anschließend wird versucht die Inhalte der Variablen *a*, *b* und *c* in die Zellen „*A1*“ bis „*A3*“ zu schreiben. Die Variable *a* wird nicht übergeben, da diese nicht deklariert ist und somit als „*Privat*“ anzusehen ist. Die Variable *b* wird nicht übergeben, da diese als „*Privat*“ deklariert ist. Einzig die Variable *c* wurde als „*Public*“ in einer „*Public Sub*“ deklariert. Der Inhalt dieser Variable wird somit vom Makro „*MitDekleration*“ wie gewünscht öffentlich zur Verfügung gestellt, so dass dieser im Makro „*abarbeiten*“ auch seine Gültigkeit hat und somit in die Zelle „*A3*“ geschrieben werden kann.

Arrays

„Normale“ Variablen, wie wir diese bisher verwendet haben, können genau einen Wert speichern. Mit Hilfe eines Arrays können mit einer Variable mehrere Werte gespeichert werden. Arrays erkennt man daran, dass dem Namen ein Klammernpaar „*( )*“ folgt, welches einen Index enthält. Angesprochen wird somit eine bestimmte Variable eines Arrays mit dem Namen der Variable und einem angefügten Index. (vgl.Das VBA-Tutorial, 2018) (vgl. Walkenbach, 2016, S. 137)

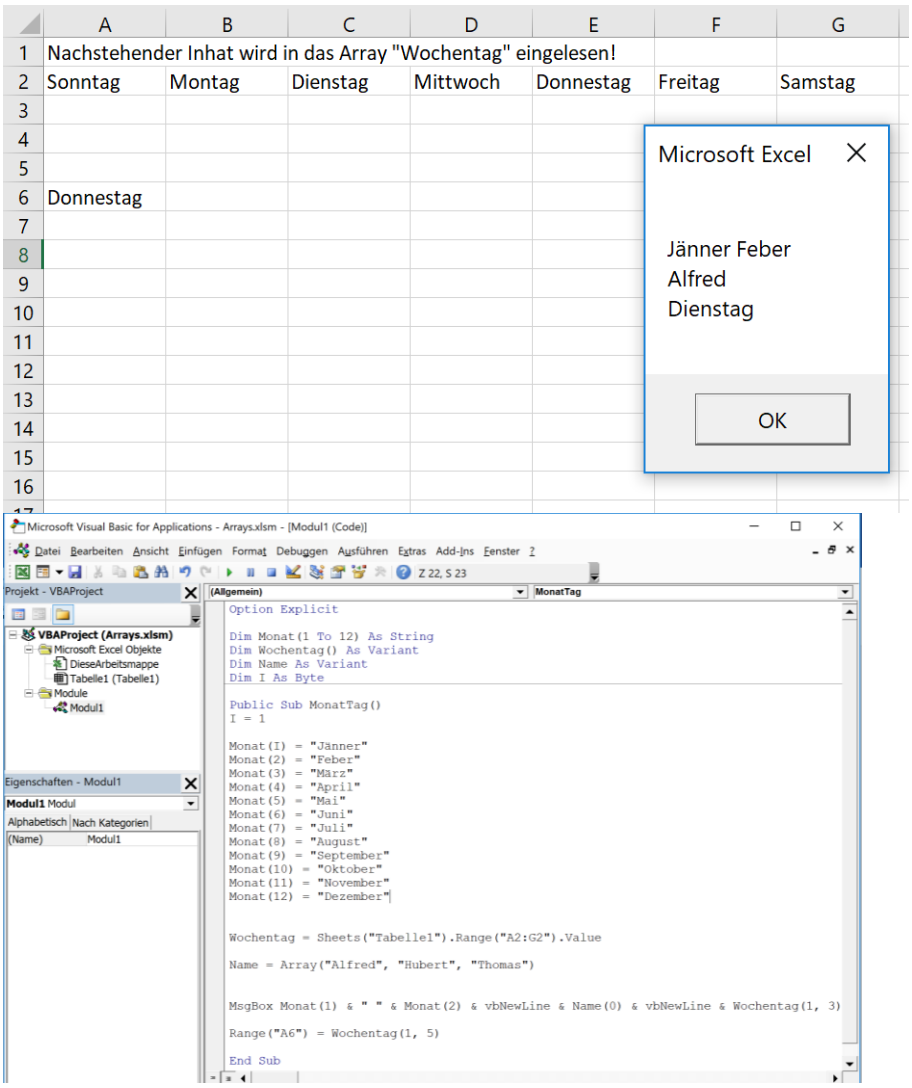


Abbildung 39: Ein- und Ausgaben aus Arrays

„Arrays“ können auf unterschiedlichste Art und Weise ein- und ausgegeben werden.

Die Variable „*Monat*“ wird aus dem Progammcode befüllt. Dem Variablennamen „Monat“ folgt der Index in den Klammern (). Dieser Index kann in Form eines nummerischen Wertes (in unserem Fall 1 bis 12) oder auch aus einer anderen Variable (in unserem Falle „*I*“, welche vorher mit dem Wert 1 belegt wurde) aufgegeben werden. Standardmäßig wird der Index ab 0 beginnend vergeben (müssten also bei 12 Werten von 0 bis 11 gehen). In der Dimensionierung wurde dieser Standard abgeändert und zwar so, dass der Index nunmehr von 1 bis 12 vergeben wurde. Derart erzeugte Arrays werden eindimensional erstellt.

Die Variable “*Name*“ wird ebenfalls über den Programmcode befüllt. Der Index wird von 0 beginnend der Reihe nach automatisch vergeben. Derart erzeugte Arrays werden eindimensional erstellt.

Die Variable „*Wochentag*“ wird über das Einlesen der Werte aus einer Excel-Tabelle befüllt. Der Index wird je Zeile und Spalte von 1 beginnend automatisch vergeben. Derartige erzeugte Arrays werden zweidimensional erstellt.

Werte aus Arrays können natürlich auch wieder in einer Excel-Tabelle ausgegeben werden.

Prozeduren und Funktionen

Prozeduren sind Sammlungen von Anweisungen, welche logisch zusammengehören und auch zusammen ausgeführt werden müssen. Durch die Aufgliederung des Programmcodes in einzelne Prozeduren wird der Programmcode übersichtlicher und leichter verständlich. Weiters können diese Funktionen auch von anderen Prozeduren verwendet werden. Der Programmcode für die in der Funktion abzuarbeitenden Befehle muss also nur einmal gewartet werden. (vgl. Theis , 2016, S. 162)

Bis jetzt haben wir mit „*Prozeduren“* gearbeitet. Diese haben folgende Struktur:

```
Sub Name(Parameter)
     Anweisungen
     Exit Sub
     Anweisungsblock
End Sub
```

„Funktionen“ unterscheiden sich von Prozeduren dadurch, dass diese Werte zurückliefern. Von der Struktur stellen sich diese wie folgt dar:

```
Funktion Name(Parameter)
     Anweisungen
     Exit Funktion
     Anweisungsblock
End Funktion
```

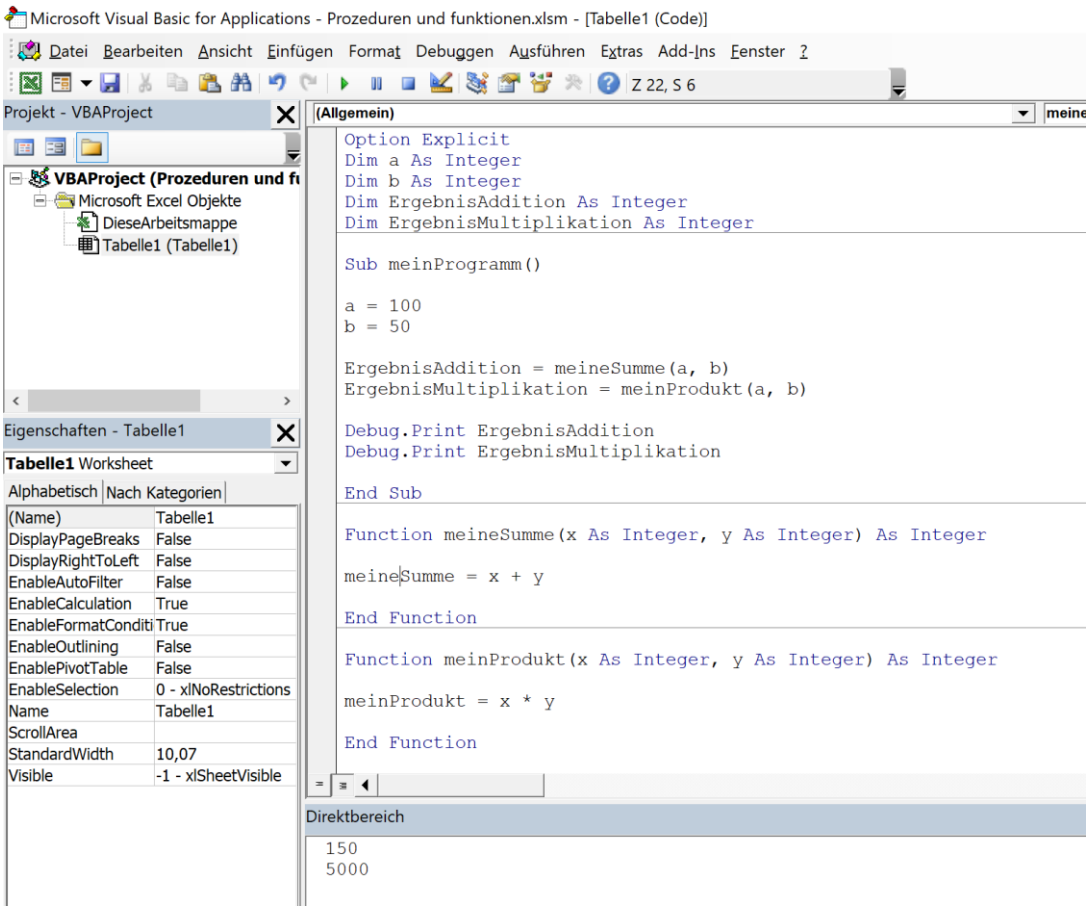


Abbildung 40: Zusammenspiel zwischen Prozeduren und Funktionen

Der vorstehende Programmcode veranschaulicht das Zusammenspiel der Prozedur „*meinProgramm*“ mit den Funktionen „*meineSumme*“ und „*meinProdukt*“. In „*meinProgamm*“ werden die Variablen „*a“* und „*b“* mit Werten belegt. Diese Werte werden als Parameter in die Funktionen „*meineSumme*“ und „*meinProdukt*“ übergeben, in welchen die eigentlichen Berechnungen durchgeführt werden. Die Ergebnisse der Berechnungen werden in die Prozedur „*meinProgramm*“ zurückgegeben. Mit den Befehlen „*Debug.Print“* wird die Ausgabe der Berechnungsergebnisse in das „*Direktfenster“* angestoßen.

Sprachelemente in VBA

Eine Programmiersprache verfügt über Sprachelemente, mit deren Hilfe Programmverzweigungen, Schleifen und sonstige Anweisungen durchgeführt werden können. Diese Sprachelemente verleihen der Programmiersprache eine Flexibilität, um Programme individuell gestalten zu können. Derartige Anweisungen können nicht mit dem Makrorekorder aufgezeichnet werden, sondern müssen manuell programmiert werden. Gute Programme entstehen durch die richtige Verwendung der zur Verfügung stehenden Sprachelemente in Verbindung mit dem dazugehörenden Vokabular (Befehle). (vgl. Held, 2016, S. 51)

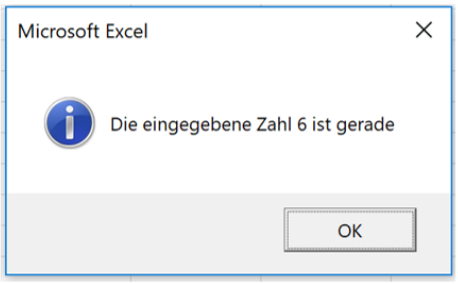
Die „If - Then – Else“ - Anweisungen

Mit Hilfe der „*If-Then-Else-Anweisungen*“ kann die Excel-Standardtabellenfunktion „*Wenn*“ in VBA abgebildet werden. Mit der „*If-Anweisung*“ wird zumeist eine Prüfung durchgeführt, wie zum Beispiel ob eine Variable einen gewissen Wert hat oder nicht. Wird die Bedingung erfüllt, dann wird die „*Then-Anweisung*“ ausgeführt. Wird die Bedingung nicht erfüllt, so wird die „Else-Anweisung“ ausgeführt insofern diese vorhanden ist (ist optional). Die „If-Befehle“ können auch geschachtelt verwendet werden, wodurch eine komplexe Entscheidungsstruktur nachgebildet werden kann. (vgl. Held, 2016, S. 52)

Die Struktur der „*If-Then-Else-Anweisungen*“ stellt sich wie folgt dar:

```
If Bedingungen Then
     Anweisungen ...
     Else
     ElseAnweisungen
End If
```

```
(Allgemein)                                                    IFAnweisung
Option Explicit
Dim Zahl As Long

Sub IFAnweisung()

Zahl = InputBox("Geben Sie eine Zahl ein: ", "Fertig")

If Zahl Mod 2 = 0 Then

    MsgBox ("Die eingegebene Zahl " & Zahl & " ist gerade"), vbInformation

    Else

    MsgBox ("Die eingegebene Zahl " & Zahl & " ist ungerade"), vbInformation

End If

End Sub
```



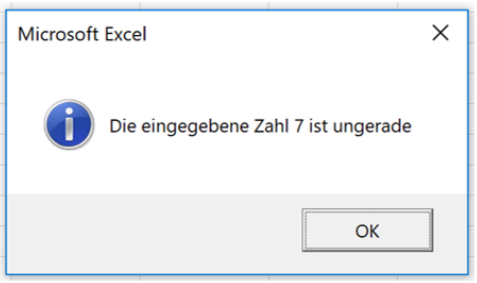


Abbildung 41: Die If-Then-Else-Anweisung samt Ausgaben

Das vorstehende Programm kann wie folgt interpretiert werden:

Der Operator Modulo (Mod) gibt den ganzzahligen Rest einer Division zweier Zahlen zurück. Ist dieser bei einer Division durch 2 gleich 0, so ist diese Zahl gerade, ansonsten muss diese ungerade sein.

Über eine „Eingabebox“ wird der Anwender aufgefordert eine Zahl einzugeben. Es wird mittels der „*If-Anweisung*“ geprüft, ob diese Zahl gerade ist oder nicht. Wenn dies zutrifft wird der „*Then*“-Befehl ausgeführt, ansonsten der „*Else*“-Befehl.

Die „Select Case“ – Anweisungen

Die „Select-Case“-Anweisung bietet die Möglichkeit um Bedingungen zu erfassen und den Programmcode übersichtlich zu erhalten. Trifft der Case-Wert zu, so wird die entsprechende Anweisung ausgeführt. Die Else-Anweisung ist optional. (vgl. Held, 2016, S. 59)

Die Struktur der *Select-Case-Anweisungen*“ stellt sich wie folgt dar:

```
Select Case Wert
Case Wert x
     Anweisungen x
Case Wert y
     Anweisungen y
Case Wert Else
     Anweisungen z
End Select
```

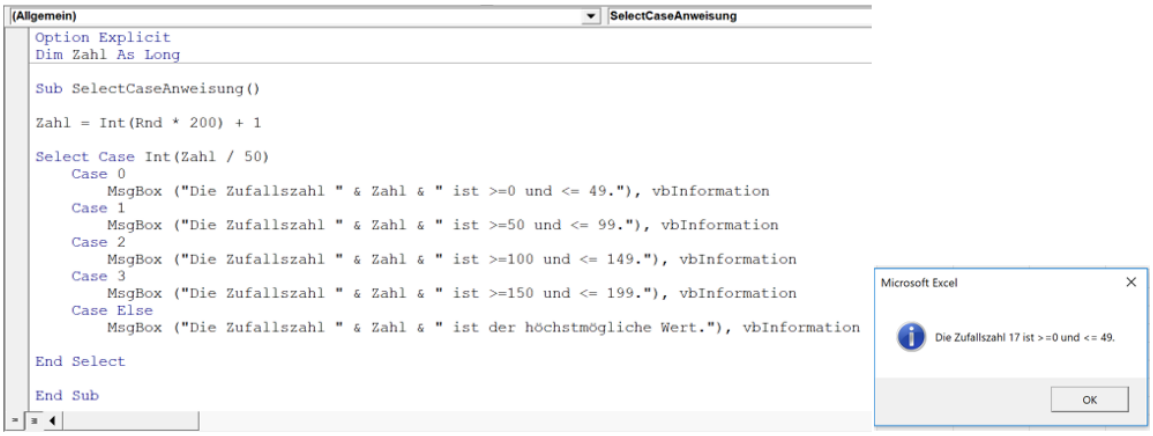


Abbildung 42: Select Case Anweisung und Ausgabe

Das vorstehende Programm kann wie folgt interpretiert werden:

Die Funktion „*Rnd*“ liefert einen Zufallswert größer gleich (>=) 0 und kleiner 1. Durch die Multiplikation mit 200 und anschließende Abrundung (Int) wird ein Wert größer gleich 0 und kleiner 200 generiert. Durch die Addition von 1 wird schlussendlich eine Zufallszahl von 1 bis inklusive 200 erzeugt.

Als Hilfsrechnung wird die so erzeugte Zufallszahl durch 50 geteilt und anschließend abgerundet. Die möglichen Ausprägungen sind somit 0, 1, 2, 3 und 4. Das sind die Werte, die mit den jeweiligen Werten der „Case“-Anweisungen verglichen werden. Ist diese mit 0, 1, 2 oder 3 ident, so wird die entsprechende Programmzeile ausgeführt. Beim Zufallswert 200 wird der „Case else“-Zweig gewählt. Dieser wird immer dann ausgeführt, wenn in den vorherigen Auswahlmöglichkeiten kein Treffer erzielt wurde.

Schleifen

Schleifen dienen dazu, um gleiche Vorgänge mehrmals hintereinander zu wiederholen. Die Schleifen werden dabei so oft durchlaufen, bis eine oder mehrere Bedingungen erfüllt werden oder nicht mehr erfüllt sind. Die Abbruchprüfung kann dabei zu Beginn oder am Ende der Schleife durchgeführt werden. (vgl. Held, 2016, S. 62f.)

Die For … Next-Schleife

Diese Schleifen werden verwendet, um eine Schleife eine bestimmte Anzahl oft durchlaufen zu lassen. Es wird dabei ein Zähler verwendet, welcher bei jedem Durchlauf erhöht oder verringert wird. In der Schleife befindet sich eine Abfrage einer Bedingung (zB durch If-Anweisung), welche dann zur Beendigung der Schleife führt. (vgl. Held, 2016, S. 63)

Die Struktur von „*For-Next*"-Schleifen sieht wie folgt aus:

```
For Zähler = Startwert To Endwert Step Schritt
     Anweisungen
     Exit For (optional)
     Anweisungen
Next Zähler
```

Die Angabe der Schritte (Step) ist optional, wird nichts angegeben, so ist dieser mit 1 belegt.

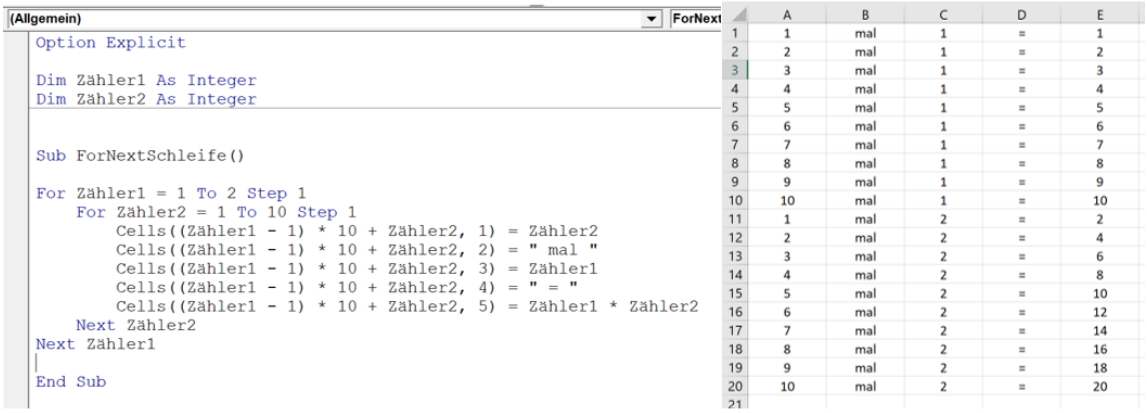


Abbildung 43: For-Next-Schleife und Ausgabe

Das vorstehende Programm kann wie folgt interpretiert werden:

Es werden zwei „*For*“-Schleifen miteinander verschachtelt. Zuerst wird die Schleife mit dem „*Zähler1“* mit dem Startwert 1 gestartet. Danach startet die Schleife mit dem „*Zähler2“* ebenfalls beim Startwert 1. Diese 2. Schleife wird nun 10 Mal durchlaufen, also so lange, bis der Endwert 10 erreicht ist. Nach dem Verlassen der inneren Schleife mit dem „*Zähler2“* wird wieder die äußere Schleife mit dem „*Zähler1*“ aktiv. Da dieser Zähler noch nicht den Endwert 2 erreicht hat, werden die weiteren Anweisungen abermals aufgerufen und somit nochmals die innere Schleife gestartet. Bei jedem Durchlauf der inneren Schleife wird eine Zeile im Excel Tabellenblatt befüllt, bis schließlich die Einer- und Zweierreihe dargestellt sind.

Mit 2 verschachtelten Schleifen kann eine zweidimensionale Matrix, welche Excel eben darstellt, abgebildet werden. Es können aber auch noch mehrere Schleifen verschachtelt werden, so gelangt man auch zu höheren Dimensionen.

Die For Each … Next-Schleifen

Diese ist die schnellste Schleife, welche Excel zur Verfügung stellt. Sie wird dazu verwendet, um Objekte in Excel (zB Zellen, Tabellen) abzuarbeiten. (vgl. Held, 2016, S. 81)

Die Struktur von „*For Each-Next*“-Schleifen sieht wie folgt aus:

```
For Each Element In Gruppe
     Anweisungen
     Exit For (optional)
     Anweisungen
Next Element
```

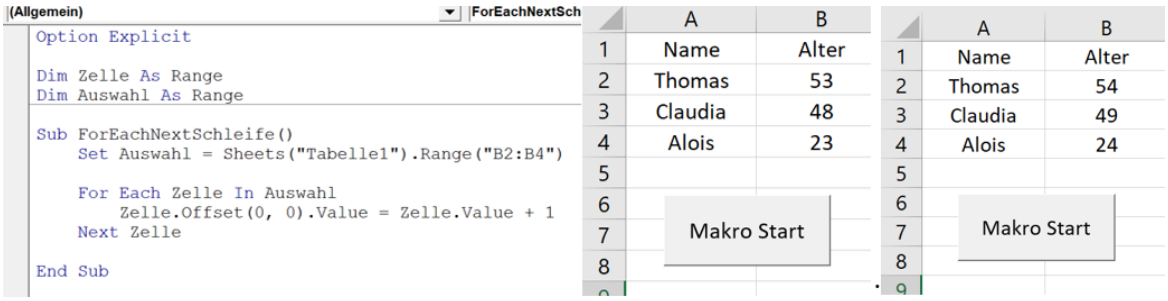


Abbildung 44: For Each-Next-Schleife und Ausgabe vor und nach dem Makrodurchlauf

Das vorstehende Programm kann wie folgt interpretiert werden:

Es wird der Bereich „B2:B4“ ausgewählt, welcher zu bearbeiten ist. Innerhalb der „*For Each*“ – Schleife wird für jedes Element des Bereiches der Programmcode ausgeführt. Im Speziellen wird jeder Wert genommen und um den Wert 1 erhöht. Die Ausgabe erfolgt durch „*Offset(0,0)*“ in die gleichen Zellen, wobei die Werte in den Klammern die Anzahl der Zeilen und Spalten angeben, um wie viele versetzt ausgegeben werden soll. Die Anweisung „*Offset(0,1)*“ würde bewirken, dass das neue Alter um eine Spalte nach rechts versetzt ausgegeben wird.

Die Do Until – Loop – Schleife vs. Do While – Loop – Schleife

Die “*Do-Until… Loop*”- Schleife wiederholt einen Anweisungsblock so lange, bis eine Bedingung erfüllt ist, wobei die Abbruchbedingungen auf verschiedenste Art (z.B. ein bestimmter Wert eines Zählers) definiert werden können. Zu beachten ist, dass die Bedingung jeweils am Ende der Schleife geprüft wird. (vgl. Held, 2016, S. 84)

Die Struktur von „*Do Until … Loop*“-Schleifen sieht wie folgt aus:

```
Do Until Bedingung
     Anweisungen
     Exit Do
Loop
```

Die “*Do-While… Loop*”- Schleife wiederholt einen Anweisungsblock so lange, bis eine Bedingung erfüllt ist, wobei die Abbruchbedingungen auf verschiedenste Art (z.B. Zähler ist kleiner als Abbruchwert) definiert werden können. Zu beachten ist, dass die Bedingung jeweils zu Beginn der Schleife geprüft wird. (vgl. Held, 2016, S. 86)

Die Struktur von „*Do While … Loop*“-Schleifen sehen wie folgt aus:

```
Do While Bedingung

     Anweisungen

     Exit Do

Loop
```

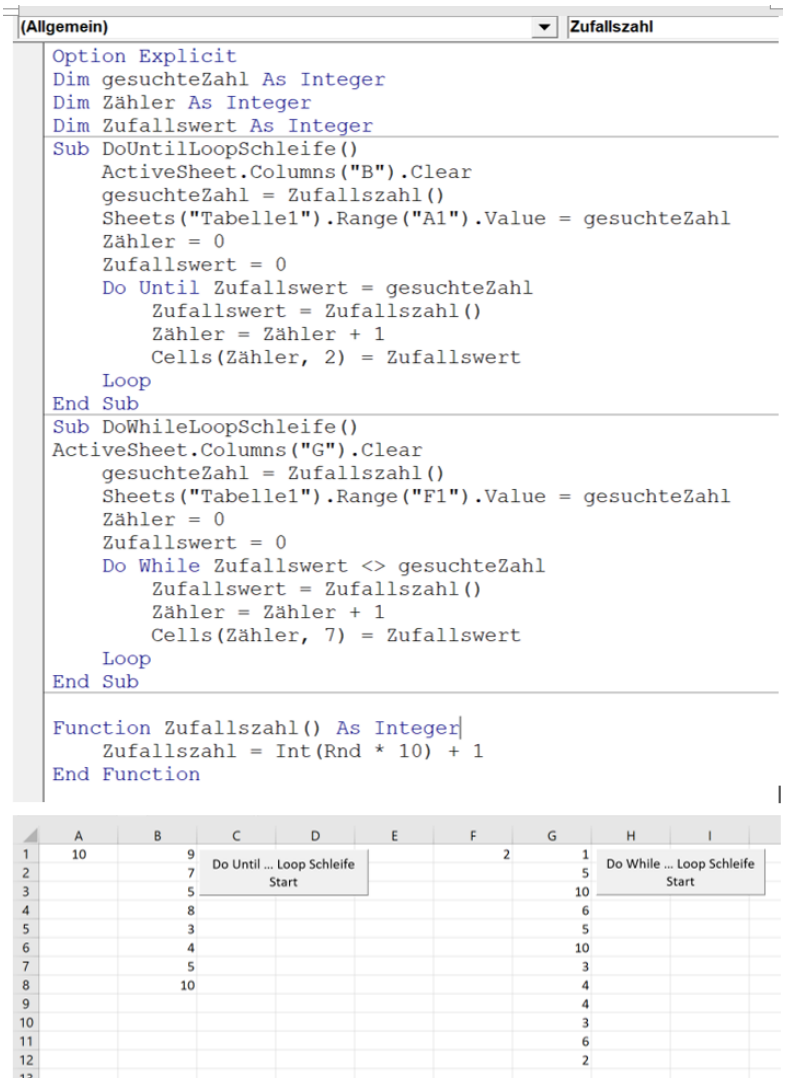


Abbildung 45: Do Until/While Loop Schleife und Ausgabe

Das vorstehende Programm kann wie folgt interpretiert werden:

Es wurden 2 Makros erstellt, welche beide die Funktion Zufallszahl verwenden. Es wird zuerst jeweils der Inhalt der Spalten B und G gelöscht (für allfällige Wiederholungen des

Makros“). Anschließend wird die Zelle „*A1*“ bzw. „*F1*“ mit einem Zufallswert belegt. Die Variablen „*Zähler*“ und „*Zufallswert*“ werden jeweils auf 0 gesetzt.

Die „*Do Until .. Loop*“-Schleife ist als „*Mache bis*“-Schleife zu verstehen. Im obigen Beispiel bedeutet dies, „Mache bis Zufallswert = gesuchterWert ist“.

Die „*Do While .. Loop*“-Schleife ist als „*Mache solange*“-Schleife zu verstehen. Im obigen Beispiel bedeutet dies, „Mache solange Zufallswert <> gesuchterWert ist“.

Die Schleife wird solange durchgeführt, bis die obige Bedingung, je nach verwendeter Schleifenart, den Wert „*Wahr*“ ergibt

- die Variable „*Zufallswert*“ mit einem Zufallswert aus der gleichnamigen Funktion belegt,
- die Variable „Zähler“ um den Wert 1 erhöht und sodann
- der Inhalt der Variable „Zufallswert“ in die nächste freie Zeile (beginnend ab 1) der Spalten „B“ bzw. „G“ geschrieben.

Es kann also mit beiden Schleifenarten, nur mit einer unterschiedlichen Formulierung der Bedingung, dasselbe Ergebnis erzielt werden.

Beispielaufgaben

Im nachstehenden Teil finden Sie einige Beispiele, welche je nach Wissensstand der Schülerinnen und Schüler als Einzelaufgaben, Gruppenaufgaben oder in Interaktion Schüler – Lehrer abgearbeitet werden können. Ausdrücklich festzuhalten ist, dass es natürlich jeweils mehrere Lösungsansätze gibt.

Es wurde versucht Beispiele zu finden, welchen einen Bezug zur Wirklichkeit haben oder auf Grund eines bestehenden Bezuges, zum Beispiel mit Mathematik, fächerübergreifend umgesetzt werden können.

Die Zahl Pi (π)

Erklärende Hinweise

Die Zahl „Pi“ spiegelt das Verhältnis eines Kreisumfanges zu seinem Durchmesser wider und stellt eine der wichtigsten Konstanten der Mathematik dar. Sie spielt aber auch im Verhältnis der Fläche eines Kreises zur Fläche seines umschließenden Quadrates eine Rolle. Die Flächen stehen im Verhältnis $\frac{\pi}{4}$: 1. (vgl. Barth, 2013, S. 30)

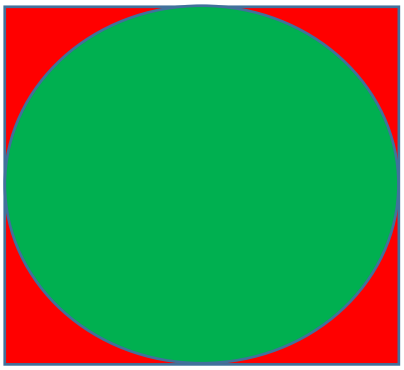


Abbildung 46: Quadrat mit eingeschriebenen Kreis

Die Fläche des Quadrates des Durchmessers ist $\frac{4}{\pi}$ mal so groß wie die Fläche des Durchmessers des Kreises.

Dies ergibt sich wie folgt: $\frac{Fläche\ Quadrat}{Fläche\ Kreis} = \frac{(2r)^2}{r^2\pi} = \frac{4r^2}{r^2\pi} = \frac{4}{\pi}$

Lässt man zufällig Tropfen auf das Quadrat regnen, so verteilen sich diese zufällig auf die Flächen innerhalb und außerhalb des Kreises. Auf Grund des vorstehenden Zusammenhanges der Flächen von Kreis und Quadrat fallen also in das Quadrat $4/\pi$ mal so viele Tropfen als in den eingeschriebenen Kreis.

Zu diesem Zweck können in einer Simulation (Monte-Carlo-Methoden), durch automatisch generierte Zufallszahlen, Punkte zufällig in einem Koordinatensystem gesetzt werden. Durch diese Methode, unter der Verwendung von Pseudo-Zufallszahlen, besteht kein Gewähr auf ein zu 100 % richtiges Ergebnis, sie ist aber sehr effizient. (vgl. Nahrstedt, 2015, S. 2f.)

Ist die Anzahl der zufällig fallenden Tropfen groß genug, so kann man sich also an die Zahl π annähern.

Durch einfache Berechnungen kann anhand der Koordinaten ermittelt werden, ob dieser Zufallspunkt innerhalb oder außerhalb des Kreises liegt.

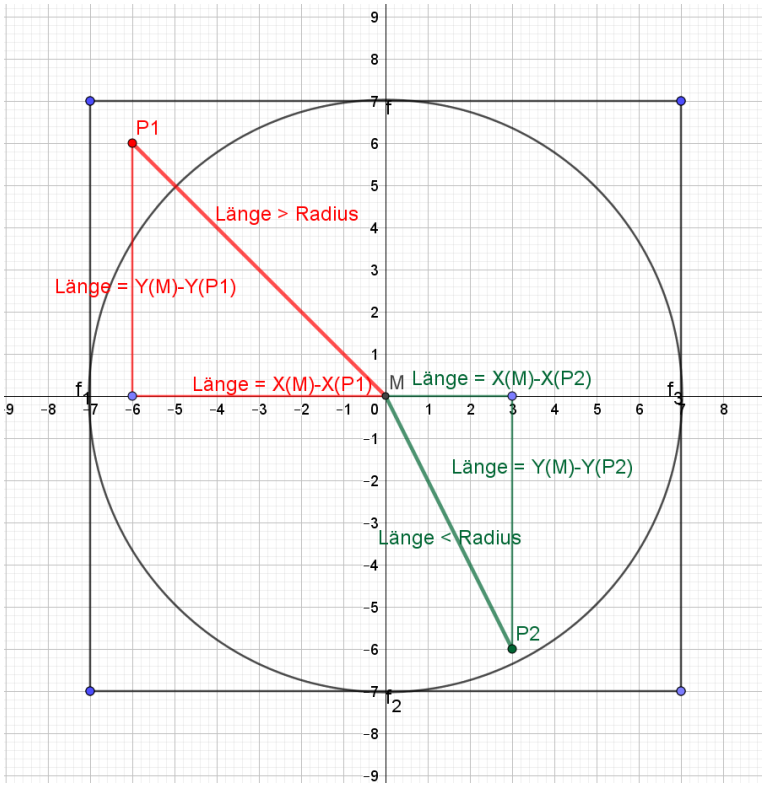


Abbildung 47: Darstellung der Berechnung der Länge des Abstandes vom Mittelpunkt

Im obigen Beispiel existieren ein Mittelpunkt M = (0/0) sowie zwei Zufallspunkte P1 = (-6/6) und P2 = (3/-6). Der jeweilige Abstand der Punkte errechnet sich über den Pythagoreischen Lehrsatz „$a^2 + b^2 = c^2$“. Dabei ergibt sich „a“ aus der Differenz der X-Koordinate des Mittelpunktes (M) und der X-Koordinate des jeweiligen Punktes „P“. Die Variable „b“ errechnet sich aus der Differenz der Y-Koordinate des Mittelpunktes (M) und der Y-Koordinate des jeweiligen Punktes „P“.

Somit kann die Länge des Abstandes eines Punktes (P) zum Mittelpunkt (M) wie folgt errechnet werden.

$$\big(X(M) - X(P)\big)^2 + \big(Y(M) - Y(P)\big)^2 = Abstand^2$$

Ist nun der Abstand^2 > Radius(K)^2 so liegt der Punkt außerhalb des Kreises, ansonsten innerhalb.

Aufgabenstellung

Entwerfen Sie auf der Basis des Zusammenhanges der Fläche eines Kreises sowie des umschreibenden Quadrates ein Simulationsprogramm, aus welchem die Zahl „*Pi*“ annäherungsweise (Monte-Carlo-Methode) ermittelt werden kann.

Die Größe des Rasters (Koordinatensystem) kann dabei selbst gewählt werden (empfohlen größer 50 kleiner 200). Wichtig dabei ist, dass eine Zelle in Excel als Pixel (Punkt) anzusehen ist.

Empfohlene mögliche Reihenfolge:

- Erstellung des Koordinatensystems (Pixel => Höhe der Zeile = 2,3 / Spaltenbreite = 0,25)
- Befüllung der einzelnen Zellen des Quadrates mit Zufallsfarben
- Erstellung eines Kreises mit Zufallsfarben (Probeberechnung der Zahl Pi)
- Erstellung des Simulationsprogrammes (Parameter Anzahl der Durchläufe, Ausgabe des Annäherungswertes zur Zahl Pi)

Musterlösung

(Allgemein) QuadratBuntFärben

```
Dim Z As Integer
Dim S As Integer
Dim innerhalbkreis As Integer
Dim ausserhalbkreis As Integer
Dim Tropfen As Integer
Dim Anzahl As Integer
Dim PI errechnet As Double

Sub QuadratBuntFärben()
Range(Cells(1, 1), Cells(200, 200)).Interior.ColorIndex = 2
For Z = 1 To 200
    For S = 1 To 200
        Cells(Z, S).Interior.ColorIndex = Int(Rnd() * 16)
    Next S
Next Z
End Sub

Sub QuadratLöschen()
Range("A1:GR200").Interior.ColorIndex = 2
End Sub

Sub VollkreisErstellen()
Range(Cells(1, 1), Cells(200, 200)).Interior.ColorIndex = 2
innerhalbkreis = 0
ausserhalbkreis = 0
For Z = 1 To 200
    For S = 1 To 200
         If (100 - S) * (100 - S) + (100 - Z) * (100 - Z) <= 10000 Then
            Cells(Z, S).Interior.ColorIndex = Int(Rnd() * 16)
            innerhalbkreis = innerhalbkreis + 1
            Cells(202, 203) = innerhalbkreis
         Else
            ausserhalbkreis = ausserhalbkreis + 1
            Cells(203, 203) = ausserhalbkreis
         End If
    PI_errechnet = innerhalbkreis / 40000 * 4
    Cells(206, 203) = PI_errechnet
    Next S
Next Z
End Sub

Sub MonteCarlo()
Tropfen = 0
innerhalbkreis = 0
ausserhalbkreis = 0

Anzahl = Cells(201, 208)
Do While Tropfen < Anzahl
    Tropfen = Tropfen + 1
    Z = Int(Rnd * 200) + 1
    S = Int(Rnd * 200) + 1

        If (100 - S) * (100 - S) + (100 - Z) * (100 - Z) <= 10000 Then
            Cells(Z, S).Interior.ColorIndex = 4 '=Farbcode grün
            innerhalbkreis = innerhalbkreis + 1
            Cells(202, 203) = innerhalbkreis
        Else
            Cells(Z, S).Interior.ColorIndex = 3 '=Farbcode rot
            ausserhalbkreis = ausserhalbkreis + 1
            Cells(203, 203) = ausserhalbkreis
        End If

    PI_errechnet = innerhalbkreis / Tropfen * 4
    Cells(206, 203) = PI_errechnet
Loop
End Sub
    PI_errechnet = innerhalbkreis / Tropfen * 4
    Cells(206, 203) = PI_errechnet
```

Abbildung 48: Musterprogramm "Die Zahl Pi"

Zusatzerläuterungen zum Musterprogramm

Die Formatierung der Zeilenhöhen und der Spaltenbreiten wäre ebenfalls über wenige VBA-Befehle möglich gewesen. Es wurde aber bewusst darauf verzichtet, da dies über Excel noch einfacher erscheint.

Beim Löschen der bestehenden Eingaben in den Prozeduren „QuadratBuntFärben“ und „QuadratLöschen“ wird veranschaulicht, dass die Zellen einer „Range“ sowohl über „Cells“, als auch über die direkten „Zelladressen“ aus Excel *("A1:GR200"),* angesprochen werden können. Der Vorteil bei der ersten Methode liegt darin, dass eine Zelle direkt im Code zur Laufzeit des Programmes über Variablen angesprochen werden kann. Diese Möglichkeit wird auch bei den darunter stehenden Prozeduren genutzt.

Bei den Prozeduren „VollkreisErstellen“ und „MonteCarlo“ wird bei der Anwendung des „Pythagoreischen Lehrsatzes“ das Quadrat einer Zahl durch deren Multiplikation dargestellt (If (100 - S) * (100 - S) + (100 - Z) * (100 - Z) <= 10000). Das Quadrat einer Zahl kann auch über die Syntax mit „^2“ (If (100 - S) ^ 2 + (100 - Z) ^ 2 <= 10000) dargestellt werden.

Im Prinzip veranschaulicht das Musterprogramm deutlich, dass ein relativ komplexer Zusammenhang mit wenigen Programmzeilen umgesetzt werden kann.

Ausgabebildschirm

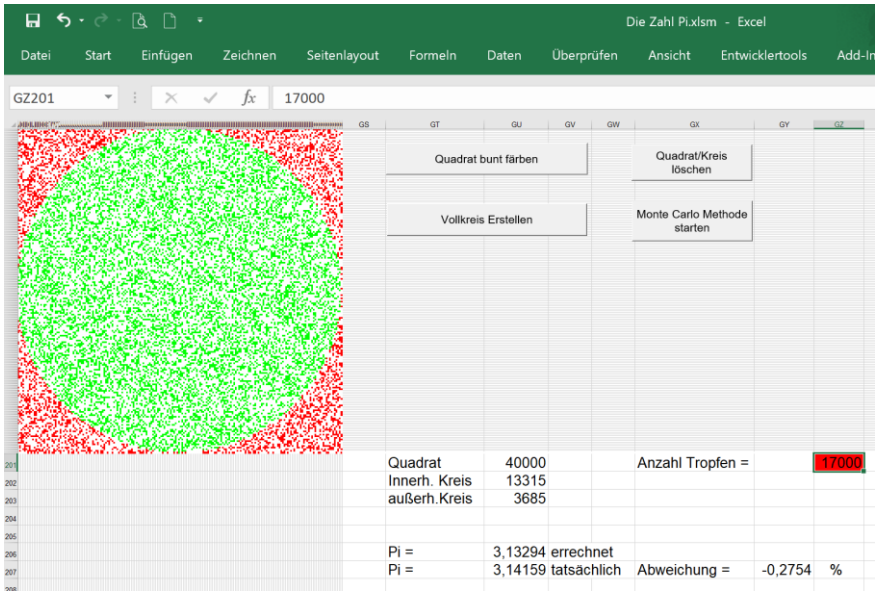


Abbildung 49: Ansicht Musterprogramm "Die Zahl Pi"

Lotto 6 aus 45

Erklärende Hinweise

Bei dem Spiel werden 6 Gewinnzahlen und eine Zusatzzahl aus einem Topf von 45 Zahlen gezogen, wobei gezogene Zahlen nicht zurückgelegt werden. Gewonnen haben Tipps ab 3 richtig getippten Zahlen oder wenn die Zusatzzahl im Tipp ist. Die Wahrscheinlichkeit auf einen „6er“ liegt bei ca. 1: 8,1 Millionen. (vgl. win2day, 2018)

Alternativ oder als Zusatz könnte auch das Euromillionen Lotto programmiert werden. Dabei werden 5 Hauptzahlen aus einem Topf von 50 Zahlen sowie 2 Sternzahlen aus einem Topf aus 12 Zahlen gezogen. Gewinne gibt es ab 2 richtigen Zahlen aus den Hauptzahlen oder einer richtigen Zahl aus dem Haupttopf und 2 richtigen Sternzahlen. Gezogene Zahlen werden nicht zurückgelegt. Die Wahrscheinlichkeit für 5+2 Richtige liegt bei ca. 1: 140 Millionen. (vgl. EuroMillions, 2018)

Aufgabenstellung

Entwerfen Sie ein Programm, mit welchem ein Spielschein mit 10 Zufallstipps befüllt wird. Weiters ist eine Lottoziehung zu simulieren, bei welcher 6 Gewinnzahlen und eine Zusatzzahl gezogen werden. Die Lottotipps und die Gewinnzahlen aus der Lottoziehung sind aus Zufallszahlen zu generieren. Die getätigten Tipps sind automatisiert mit den gezogenen Gewinnzahlen zu vergleichen und die Anzahl der richtigen Tipps zu ermitteln. Ebenso ist zu ermitteln, ob die Zusatzzahl richtig getippt wurde. Bei der Ausprogrammierung ist darauf zu achten, weil die Ziehung ohne Zurücklegen der gezogenen Zahlen erfolgt, dass keine doppelten Zahlen pro Lottotipp, oder bei der Ziehung der Gewinnzahlen, generiert werden.

Empfohlene mögliche Reihenfolge:

- Erarbeitung eines Algorithmus zur Ermittlung von Zufallszahlen unter Vermeidung von Doppelausgaben.
- Erstellung des Feldes zur Anzeige der generierten 10 Lottotipps sowie eines Bereiches für die Darstellung der bei der Simulation der Lottoziehung ermittelten Gewinnzahlen.
- Ausarbeitung eines Konzeptes zur Darstellung der Gewinnzahlen (z.B. farbliche Hervorhebung).
- Erstellung eines Bereiches zur Ausgabe des Ergebnisses der Gewinnermittlung
- Ausprogrammierung der Gewinnermittlung und Anzeige von Gewinnen.
- Erstellung einer Löschprozedur für die im Zuge der Gewinnermittlung getätigten Ausgaben und farblichen Darstellungen, sowie zur Rücksetzung aller Daten.

Musterlösung

(Allgemein) | LottoscheinAusfüllen

```
Option Explicit
Public Lottozahl As Byte
Dim AlleZahlen(1 To 45)
Dim Auswahl As Range
Dim Zelle As Range
Dim Zähler1 As Integer
Dim Zähler2 As Integer
Dim Zähler3 As Integer
Dim Richtige As Integer
Dim clear As Byte

Sub Lottoziehung()

For Zähler1 = 1 To 45
    AlleZahlen(Zähler1) = 0
Next Zähler1
Set Auswahl = Sheets("Tabelle1").Range("C14:I14")
For Each Zelle In Auswahl
    Do
        Lottozahl = Zufallszahl()
        If AlleZahlen(Lottozahl) <> 1 Then
            Zelle.Offset(0, 0).Value = Lottozahl
            AlleZahlen(Lottozahl) = 1
            Exit Do
        End If
    Loop
Next Zelle
Call Löschen(1)

End Sub
Sub LottoscheinAusfüllen()

Call Löschen(1)

For Zähler1 = 1 To 10
    For Zähler2 = 1 To 45
        AlleZahlen(Zähler2) = 0
    Next Zähler2
    Cells(2 - 1 + Zähler1, 3) = Zähler1
    For Zähler2 = 1 To 6
        Do
            Lottozahl = Zufallszahl()
            If AlleZahlen(Lottozahl) <> 1 Then
               Cells(2 - 1 + Zähler1, 3 + Zähler2) = Lottozahl
               AlleZahlen(Lottozahl) = 1
               Exit Do
           End If
       Loop
    Next Zähler2

Next Zähler1
End Sub

Sub Gewinnabfrage()

Call Löschen(1)

For Zähler1 = 1 To 10   'Lottotipps
    For Zähler2 = 1 To 6    'Zahl 1 bis 6 des Lottotips
        For Zähler3 = 1 To 7    'Gezogene zahlen
            If Cells(2 - 1 + Zähler1, 4 - 1 + Zähler2) = Cells(14, 3 - 1 + Zähler3) Then
                If Zähler3 < 7 Then
                    Richtige = Richtige + 1
                    Cells(2 - 1 + Zähler1, 4 - 1 + Zähler2).Interior.ColorIndex = 10
                Else
                    Cells(1 + Zähler1, 12) = "Ja"
                    Cells(2 - 1 + Zähler1, 4 - 1 + Zähler2).Interior.ColorIndex = 6
                End If
            End If
        Next Zähler3
    Next Zähler2
    Cells(1 + Zähler1, 11) = Richtige
    Richtige = 0
```

Abbildung 50: Musterprogramm Lotto 6 aus 45 Teil 1

```
Next Zähler1

End Sub

Function Zufallszahl() As Byte

Zufallszahl = Int(Rnd * 45) + 1 'Zufallszahl zwischen 1 und 45

End Function

Sub Löschen(clear)

If clear = 1 Then
    For Zähler1 = 1 To 10
        For Zähler2 = 1 To 6
            Cells(2 - 1 + Zähler1, 4 - 1 + Zähler2).Interior.ColorIndex = 2
        Next
    Cells(2 - 1 + Zähler1, 12) = ""
    Cells(2 - 1 + Zähler1, 11) = ""
    Next
End If

If clear = 2 Then
    For Zähler1 = 1 To 10
        For Zähler2 = 1 To 6
            Cells(2 - 1 + Zähler1, 4 - 1 + Zähler2) = ""
        Next
    Next
End If

If clear = 3 Then
    For Zähler1 = 1 To 7
         Cells(14, 3 - 1 + Zähler1) = ""
    Next
End If
End Sub

Sub BildschirmLöschen()

Call Löschen(1)
Call Löschen(2)
Call Löschen(3)

End Sub
```

Abbildung 51: Musterprogramm Lotto 6 aus 45 Teil 2

Zusatzerläuterungen zum Musterprogramm

Zur Ermittlung allfälliger doppelter Zahlen bei der Generierung des Quick-Tipps sowie bei der Gewinnzahlenermittlung stehen mehrere Möglichkeiten zur Auswahl. In diesem Fall wird ein eigenes Array „AlleZahlen“ mit 45 Zahlen erstellt, wobei alle Werte mit „0“ belegt werden. Bei der Generierung einer Zahl wird der entsprechende Eintrag im Array auf „1“ gesetzt. Bei allen weiteren Zufallszahlengenerierungen wird sodann abgefragt, ob dieser Wert „0“ ist. Ist dies der Fall, so wurde die entsprechende Zahl noch nicht gezogen.

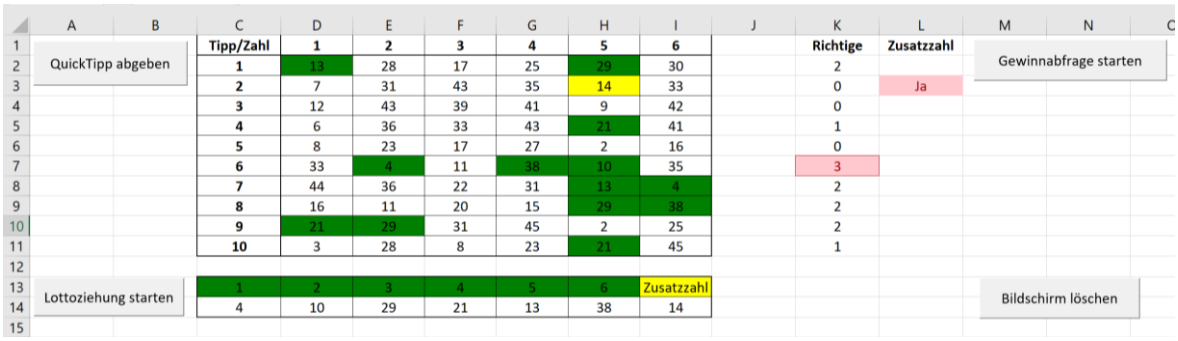
Bei der Gewinnabfrage wäre auch eine Ausprogrammierung über „For Each“-Schleifen, unter Verwendung von Arrays oder „Do While“-Schleifen möglich. Durch die gewählte Vorgehensweise über die Verwendung der „For-Next“-Schleifen wird die Arbeitsweise bei verschachtelten Schleifen meiner Meinung nach aber sehr gut veranschaulicht.

Die Prozedur „*Löschen*“ wäre auch mit der „Case“-Anweisung möglich gewesen.

Weiters besteht durch die weitgehende Programmierung über VBA und die Aufteilung in mehrere Prozeduren die Möglichkeit, Zahlen sowohl bei den Tipps als auch bei den Gewinnzahlen manuell zu erfassen und dann die Gewinnabfrage durchzuführen. Als Erweiterung für die Aufgabe könnte dabei die Prüfung auf allfällige Falscheingaben (Zahl kleiner 1 oder größer 45) oder Doppeleingaben sein.

Die farbliche Hervorhebung der Gewinnzahlen bei den Quick-Tipps wurde über VBA umgesetzt. Die farbliche Hervorhebung der Tipps mit 3 oder mehreren richtigen Zahlen in der Spalte „*K*“ und bei der richtigen Zusatzzahl der Spalte „*L*“ wurde über die bedingte Formatierung in Excel umgesetzt. Es soll damit veranschaulicht werden, dass sowohl über VBA als auch über Excel zufriedenstellende Lösungen möglich sind. In der Ausprogrammierung muss jeder für sich selbst entscheiden, welchen Weg man für machbar und effektiv hält.

Ausgabebildschirm



Tipp/Zahl	1	2	3	4	5	6	Richtige	Zusatzzahl
1	13	28	17	25	29	30	2	
2	7	31	43	35	14	33	0	Ja
3	12	43	39	41	9	42	0	
4	6	36	33	43	21	41	1	
5	8	23	17	27	2	16	0	
6	33	4	11	38	10	35	3	
7	44	36	22	31	13	4	2	
8	16	11	20	15	29	38	2	
9	21	29	31	45	2	25	2	
10	3	28	8	23	21	45	1	

1	2	3	4	5	6	Zusatzzahl
4	10	29	21	13	38	14

Abbildung 52: Ansicht Musterprogramm "Lotto 6 aus 45"

Der einarmige Bandit

Erklärende Hinweise

Es handelt sich dabei wohl um den berühmtesten Glückspielautomat, welcher gegen Ende des 19. Jahrhunderts in den USA erfunden wurde. Diese Automaten prägen das Bild vieler Casinos und haben den Ruf, dass sie den Spielern das Geld aus der Tasche ziehen. Die Maschinen bestehen aus Walzen (ursprünglich 3) mit verschiedenen Symbolen. Die Walzen wurden nach Einwurf einer Münze in Rotation versetzt. Die Walzen kommen selbstständig zum Stillstand und je nach Stellung der zu sehenden Symbole zahlt die Maschine automatisch den gewonnenen Betrag aus. (vgl. Schmitz, 2014)

Aufgabenstellung

Entwerfen Sie ein Programm, mit welchem ein „Einarmiger Bandit“ (Slot-Maschine) nachgebildet wird. Die Maschine soll mindestens 3 Walzen besitzen, welche nach einem Zufallsalgorithmus zum Stillstand kommen und ein Zufallsbild zeigen sollen. Gewinne sollen selbstständig aus einer Gewinntabelle ausgelesen und automatisch einem Konto (Credit) gutgeschrieben werden. Es soll dem Spieler die Möglichkeit gegeben werden die Höhe seiner Einsätze selbst zu wählen. Bei der Ausprogrammierung sollte darauf Bedacht genommen werden, dass die Gewinnwahrscheinlichkeiten und Auszahlungsquoten so zu wählen sind, dass auf Dauer der Aufsteller des „einarmigen Banditen“ Gewinne erzielt.

Empfohlene mögliche Reihenfolge:

- Überlegung wie viele Walzen (3 oder 4) mit wie vielen Symbolen (kleiner gleich 7) darauf verwendet werden.
- Überlegung sinnvoller Gewinnwahrscheinlichkeiten
- Schaffung eines Spielfeldes und eines Bereiches für Parameter
- Ausprogrammierung zuerst einer und dann mehrerer Rollen, welche selbstständig nach einem Zufallsprinzip zum Stillstand kommen.
- Programmierung der Gewinnermittlung durch Vergleich des Bildes mit den zum Stillstand gekommenen Rollen mit einer Gewinntabelle und automatische Auszahlung der Gewinne auf das Spielerkonto „Credit“

Musterlösung

„*Modul 1*“

(Allgemein) (Deklarationen)

```
Dim ZählerRolle
Public WertRolle1 As Integer
Public WertRolle2 As Integer
Public WertRolle3 As Integer
Public WertRolle4 As Integer
Dim break1 As Integer
Dim break2 As Integer
Dim break3 As Integer
Dim break4 As Integer
Public Kredit As Integer
Public Einsatz As Integer
Public Gewinn As Integer

Sub EinarmigerBandit()
Tabelle1.CommandButton1.Enabled = False

Kredit = Tabelle1.Cells(1, 7)
Einsatz = Tabelle1.Cells(1, 8)
Kredit = Kredit - Einsatz
Cells(1, 7) = Kredit
Gewinn = 0

ZählerRolle = 0
break1 = Int(Rnd * 10) + 8
break2 = Int(Rnd * 10) + 6
break3 = Int(Rnd * 10) + 4
break4 = Int(Rnd * 10) + 2

Do While ZählerRolle < (break1 + break2 + break3 + break4)
    ZählerRolle = ZählerRolle + 1
    If ZählerRolle < break1 Then
        WertRolle1 = Int(Rnd * 15) + 1
        Cells(2, 4) = WertRolle1
        Cells(1, 4) = Tabelle2.Cells(WertRolle1, 3)
    End If
    If ZählerRolle < break1 + break2 Then
      WertRolle2 = Int(Rnd * 15) + 1
      Cells(2, 3) = WertRolle2
     Cells(1, 3) = Tabelle2.Cells(WertRolle2, 3)
    End If
    If ZählerRolle < break1 + break2 + break3 Then
        WertRolle3 = Int(Rnd * 15) + 1
        Cells(2, 2) = WertRolle3
        Cells(1, 2) = Tabelle2.Cells(WertRolle3, 3)
    End If
    If ZählerRolle < break1 + break2 + break3 + break4 Then
      WertRolle4 = Int(Rnd * 15) + 1
      Cells(2, 1) = WertRolle4
      Cells(1, 1) = Tabelle2.Cells(WertRolle4, 3)
    End If
    Warten (100)
Loop
Gewinnermittlung (Gewinn)
Tabelle1.CommandButton1.Enabled = True
End Sub

Public Sub Warten(ByVal MilliSekunden As Double)

 Dim i As Double
 Dim Ende As Double

 Ende = Timer + (MilliSekunden / 1000)

 Do While i < Ende
   DoEvents
   i = Timer
 Loop

End Sub
```

Abbildung 53: Musterprogramm "Der einarmige Bandit" Teil 1

```
Function Gewinnermittlung(Gewinn)

Gewinnzahl1 = Tabelle2.Cells(WertRolle1, 2)
Gewinnzahl2 = Tabelle2.Cells(WertRolle1, 2) + Tabelle2.Cells(WertRolle2, 2) * 10
Gewinnzahl3 = Tabelle2.Cells(WertRolle1, 2) + Tabelle2.Cells(WertRolle2, 2) * 10 +
              Tabelle2.Cells(WertRolle3, 2) * 100
Gewinnzahl4 = Tabelle2.Cells(WertRolle1, 2) + Tabelle2.Cells(WertRolle2, 2) * 10 +
              Tabelle2.Cells(WertRolle3, 2) * 100 + Tabelle2.Cells(WertRolle4, 2) * 1000

Select Case Gewinnzahl4
   Case 4444
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(15, 6)
        GoTo 10
   Case 3333
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(13, 6)
        GoTo 10
   Case 2222
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(10, 6)
        GoTo 10
   Case 1111
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(7, 6)
        GoTo 10
End Select

Select Case Gewinnzahl3
   Case 444
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(14, 6)
        GoTo 10
   Case 333
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(12, 6)
        GoTo 10
   Case 222
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(9, 6)
        GoTo 10
   Case 111
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(6, 6)
        GoTo 10
End Select

Select Case Gewinnzahl2
   Case 33
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(11, 6)
        GoTo 10
   Case 22
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(8, 6)
        GoTo 10
   Case 11
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(5, 6)
        GoTo 10
End Select

Select Case Gewinnzahl1
   Case 1
        Gewinn = Tabelle1.Cells(1, 8) * Tabelle1.Cells(4, 6)
        GoTo 10
End Select
10 If Gewinn > 0 Then
        Cells(1, 9) = Gewinn
        Warten (2000)
            For i = 1 To Gewinn
                Kredit = Kredit + 1
                Cells(1, 7) = Kredit
                Cells(1, 9) = Cells(1, 9) - 1
                If Gewinn > 100 Then
                    Warten (25)
                Else:
                    Warten (50)
                End If
            Next i
        Cells(1, 9) = ""

End If

End Function
```

Abbildung 54: Musterprogramm "Der einarmige Bandit" Teil 2

„*Tabelle 1 (CommandButton1)*“

```
CommandButton1                                    Click
Option Explicit

Private Sub CommandButton1_Click()

Call EinarmigerBandit

End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

End Sub
```

Abbildung 55: Musterprogramm "Der einarmige Bandit" Teil 3

Zusatzerläuterungen zum Musterprogramm

Bei der Ausprogrammierung wurde die Darstellung der Parametrierung und der Gewinntabelle in Excel abgebildet. Dies einerseits um aufzuzeigen, wie gut Excel und VBA miteinander kombiniert werden können und andererseits, um den Programmcode nicht zu einem unübersichtlichen Ausmaß anwachsen zu lassen. Eine Neuheit ist auch, dass der Programmcode nicht zur Gänze im Modul 1 geschrieben ist.

In der Tabelle1 sind 2 Buttons „Spiel starten“ angebracht. Dazu ist zu sagen, dass im Prinzip 2 Arten von Steuerelementen verwendet werden können, die normalen Steuerelemente und die ActiveX-Steuerelemente.

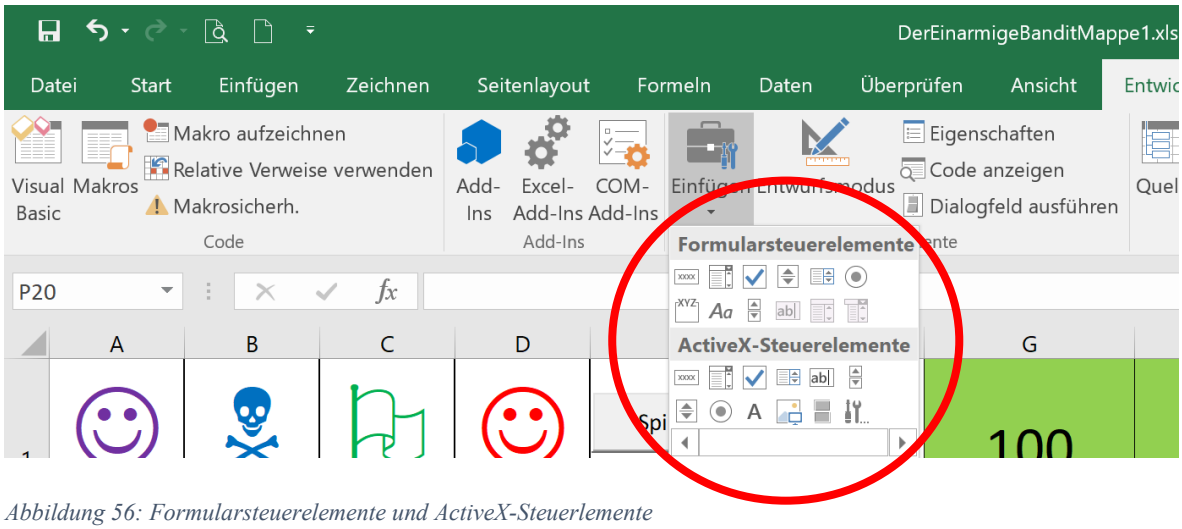


Abbildung 56: Formularsteuerelemente und ActiveX-Steuerlemente

ActiveX-Steuerelemente bieten auch die Möglichkeit den Status eines Steuerelementes zur Laufzeit des Programmes zu verändern.

Beim unteren Button handelt es sich um ein wie bisher verwendetes Formularsteuerelement, während der obere Button ein ActiveX-Steuerelement ist. Beim unteren Button ist das Makro „*EinarmigerBandit*“ zugewiesen (Rechte Maustaste Befehl „*Makro zuweisen*“.

Beim oberen ActiveX Steuerelement wird unter der „*Tabelle1*“ (*ProjektEditor*) eine eigene Prozedur angelegt. Im vorliegenden Fall besteht unter dem Ereignis „Click“ eine Prozedur, aus welcher heraus mit dem Befehl „*Call EinarmigerBandit*“ das entsprechende Makro aufgerufen wird. Der Vorteil liegt darin, dass in der Hauptprozedur unter „Modul1“ dieser Button mit dem Befehl „*Tabelle1.CommandButton1.Enabled = False*“ auf inaktiv gestellt werden kann. Das bedeutet, dass während des Ablaufes der Prozedur „*EinarmigerBandit*“ dieser kein zweites Mal aktiviert werden kann. Der Button wird erst nach dem kompletten Durchlauf mit dem Befehl „*Tabelle1.CommandButton1.Enabled = True*“ wieder aktiviert. Der Grund liegt darin, dass das Programm im Falle einer Gewinnauszahlung und dem gleichzeitigen neuerlichen Start der Prozedur fehlerhaft arbeitet. Dies kann durch Betätigung des unteren Buttons während der Gewinnauszahlung ausprobiert werden.

Bei der Prozedur „Einarmiger Bandit“ handelt es sich um das Programm, welches den Ablauf eines Spieles steuert. Im ersten Block wird zuerst der Wert des bestehenden Guthabens (Kredit), sowie die Höhe des Einsatzes pro Spiel aus der Exceltabelle ausgelesen. Der Kredit wird um den Wert des Einsatzes vermindert und wieder in die Exceltabelle zurückgeschrieben.
Im Folgenden werden Zufallszahlen für den Stopp der 4 Rollen ermittelt, welche in der anschließenden Schleife abgefragt werden. Je nach Höhe des Zählers der Durchläufe werden die einzelnen Rollen der Reihe nach gestoppt. Die Werte der Rollen werden im Bereich „*A2*“ bis „*D2*“ der Exceltabelle angezeigt. Diese Anzeige wird nur zum Zwecke der Veranschaulichung ausgegeben. Auf Basis dieser Rollenwerte werden die einzelnen Werte aus der Excel-Tabelle „Parameter“ (dem Rollenwert entsprechende Zeile der Spalte C) geholt und in den Bereich „*A1*“ bis „*A4*“ der Excel „Tabelle1“ übernommen. Durch die Formatierung der Schrift auf den Typ „SymbolPS“ werden die Symbole der Rollen dargestellt.

Da der Rechner schneller wäre als es für einen ansprechenden Programmablauf gut ist, werden Pausen im Ablauf eingelegt. Dies erfolgt über den Aufruf der Funktion „*Warten*“

wobei als Parameter Millisekunden übergeben werden. Nach Rückkehr aus dieser Prozedur wird das Programm weiter abgearbeitet.

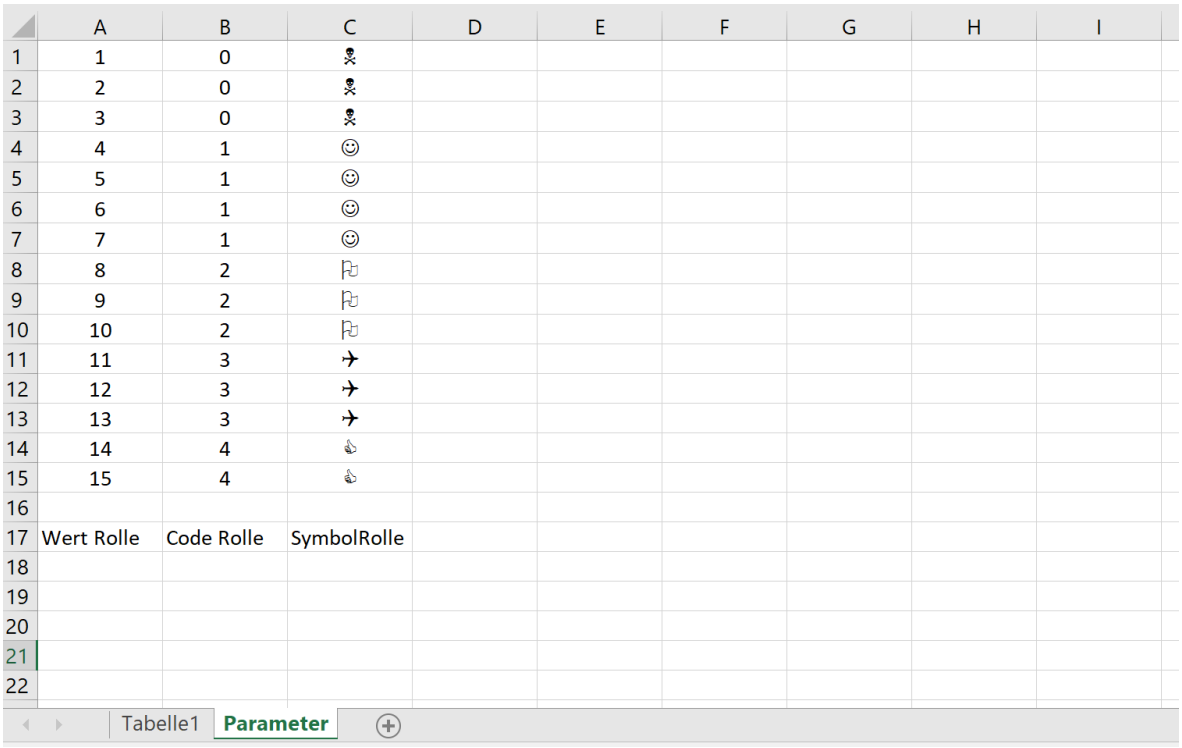
Wenn alle Rollen gestoppt sind und nach der letzten Rückkehr aus der Prozedur „*Warten*“ wird die Funktion „*Gewinnermittlung*“ aufgerufen. Nach dieser Abarbeitung wird der ActiveX-Button „Spiel starten“ wieder aktiviert und das nächste Spiel kann gestartet werden.

Die Prozedur „*Warten*“ übernimmt den in Millisekunden übergebenen Wert und errechnet aus der aktuellen Zeit (Timer) und der Wartezeit in Millisekunden einen Endwert der Wartezeit. Die folgende Schleife wird so lange durchlaufen, bis der Endwert erreicht ist. Danach wechselt der Ablauf wieder zurück zur entsprechenden Stelle der Prozedur „*Einarmiger-Bandit*“.

Die Prozedur „*Gewinnermittlung*“ ermittelt zunächst aus den im Bereich „*A2*“ bis „*D2*“ der „*Tabelle1*“ ausgegebenen Rollenwerte, in Verbindung mit dem aus der Tabelle „*Parameter*“ abzulesenden Rollencode (Saplte B), vier Gewinnzahlen, mit der Einerstelle, danach der Einer- und der Zehnerstelle usw. Im folgenden „*Select Case*“-Block werden die entsprechend der Gewinntabelle möglichen Gewinnzahlen abgeglichen und im Falle der Übereinstimmung der Gewinn errechnet. Dies erfolgt durch Multiplikation des Einsatzes (Zelle „*H1*“ der „*Tabelle1*“) mit dem entsprechenden Multiplikator je Gewinn (Bereich F4 bis F15 der Tabelle1). Im folgenden „*If*“-Block und der darin enthaltenen „*For*“-Schleife wird der Gewinn heruntergezählt und gleichzeitig der Guthaben (Kredit) erhöht.

	A	B	C	D	E	F	G	H	I
1					Spiel Start		99	1	1
2	5	3	9	4			Kredit	Einsatz	Gewinn
3	Gwinntabelle				Wahrscheinlichkeit	Einstaz mal			
4					1/3	1			
5					1/14	3			
6					1/52	10			
7					1/197	50			
8					1/25	5			
9					1/125	30			
10					1/625	125			
11					1/25	5			
12					1/125	30			
13					1/625	100			
14					1/421	125			
15					1/3164	500			
16									
17									
18					Spiel Start				
19									
20									

Abbildung 57: Musterprogramm "Der einarmige Bandit" Spielfeld



	A	B	C	D	E	F	G	H	I
1	1	0	☠						
2	2	0	☠						
3	3	0	☠						
4	4	1	☺						
5	5	1	☺						
6	6	1	☺						
7	7	1	☺						
8	8	2	⚐						
9	9	2	⚐						
10	10	2	⚐						
11	11	3	✈						
12	12	3	✈						
13	13	3	✈						
14	14	4	✌						
15	15	4	✌						
16									
17	Wert Rolle	Code Rolle	SymbolRolle						
18									
19									
20									
21									
22									

Tabelle1 **Parameter**

Abbildung 58: Ansicht Musterprogramm "Der einarmige Bandit" Parameterblatt

Eine mögliche Erweiterung des Programmes wäre ein Zusatz „Gamble“ im Falle eines Gewinnes. Gewinne werden dann nicht mehr automatisch dem Guthaben (Kredit) zugeschrieben, sondern die Spielerin oder der Spieler hat die Möglichkeit den Gewinn zu verdoppeln, wenn das „Gamble“ gewonnen wird. Die Wahrscheinlichkeit dafür sollte etwas 50 % (zum Beispiel 49:51) liegen. Verliert die Spielerin oder der Spieler das „Gamble“, so ist der zuletzt erspielte Gewinn wieder verloren und wird nicht dem Guthaben (Kredit) gutgeschrieben.

Literaturverzeichnis

Barth, A. P. (2013). *Algorithmik für Einsteiger.* Baden, Schweiz: Springer Stektrum.

Buhl, A. P., & Strauch , P. P. (2005). *Grundkurs VBA: Einführung in die Programmentwicklung mit Visual Basic for Applications in Excel.* München: Oldenbourg Wissenschaftsverlag GmbH.

Das VBA-Tutorial. (16. 02 2018). Abgerufen am 16. Feber 2018 von http://www.vba-tutorial.de/variablen/datentypen.htm

EuroMillions. (23. Feber 2018). Abgerufen am 23. Feber 2018 von https://www.euro-millions.com/de/regeln

Held, B. (2016). *Richtig einsteigen: Excel VBA-Programmierung.* dpunkt.verlag GmbH.

Nahrstedt, H. (2015). *Die Monte-Carlo-Methode.* Möhnesee: Springer Vieweg.

Schmitz, A. (02. 05 2014). *Planet Wissen*. (Westdeutscher Rundfunk Köln) Abgerufen am 25. 02 2018 von https://www.planet-wissen.de/kultur/metropolen/las_vegas_spielparadies_in_der_wueste/pwiedereinarmigebandit100.html

Theis , T. (2016). *Einstieg in VBA mit EXCEL* (4. aktualisierte Aufllage 2016 Ausg.). Bonn: Rheinwerk Verlag GmbH.

Walkenbach, J. (2016). *Excel-VBA für Dummies.* (J. Muhr, Übers.) Weinheim, Deutschland: WILEY-VCH Verlag GmbH & Co KG.

win2day. (23. Feber 2018). Abgerufen am 23. Febr 2018 von https://www.win2day.at/lotterie/lotto

Abbildungsverzeichnis