

Unterlagen für das obligatorische Fach Informatik

Jacques Mock Schindler

02.10.2025

Inhaltsverzeichnis

Willkommen	5
Programm	5
Materielle Voraussetzungen	5
Beurteilung	6
I. Anleitungen	7
1. Speicherorganisation	8
1.1. Dateinamen und Pfade	8
1.2. Dateien in der Cloud	9
1.3. Dateistruktur für die Schule	9
2. Arbeitsumgebung (Arbeiten mit Jupyter Notebooks)	11
2.1. Ausgangslage	11
2.2. Installation von Python	11
2.3. Hello World	12
2.4. Arbeitsumgebung	13
2.5. Öffnen bestehender Jupyter Notebooks	18
2.6. Häufige Fehlermeldungen	19
II. Einführung in Python	22
3. Problemlösung in der Informatik	23
3.1. Ausgangslage	23
3.2. Beispiel: Tricolore	24
3.2.1. Rechtecke zeichnen	24
3.3. Beispiel: Österreichische Flagge	26
3.4. Beispiel: Schweizerfahne	26
3.5. Beispiel: Tessiner Wappen	26
3.6. Musterlösungen	26
3.6.1. Trikolore	26
3.6.2. Österreichische Flagge	27
3.6.3. Schweizerfahne	27
3.6.4. Tessiner Wappen	27

4. Variablen in Python	29
4.1. Vorbemerkungen: Python als Rechner	29
4.2. Variablen	29
4.3. Funktionen in Python	31
4.4. Datentypen	31
4.4.1. Integer	32
4.4.2. Gleitkommazahl	33
4.4.3. Komplexe Zahlen	33
4.4.4. String	34
4.4.5. Boolean Type	34
4.5. Funktionen mit Type-Hints	35
4.6. Ausgewählte Musterlösungen	36
4.6.1. Funktion zur Multiplikation zweier Zahlen	36
4.6.2. Gleitkommazahlen in wissenschaftlicher Schreibweise	36
5. Wiederholungen in Python (For-Loops)	37
5.1. Anwendungsübung zu Wiederholungen in Python	38
5.1.1. Schritt 1: Zerlege das Bild in seine Einzelteile	39
5.1.2. Schritt 2: Positioniere die Blütenblätter	40
5.1.3. Schritt 3: Zeichnen aller erforderlichen Blütenblätter	40
5.1.4. Schritt 4: Kombiniere alle Blütenblätter	41
5.1.5. Schritt 5: Scheibe im Zentrum der Blume	44
5.1.6. Schritt 6: Kombination der Elemente zur vollständigen Blume	45
6. Programmverzweigungen (Bedingungen)	46
6.1. Anwendungsübung zu Bedingungen in Python	48
6.1.1. Aufgabenstellung	48
6.1.2. Musterlösung	49
7. Programmieren einfacher Funktionen	51
7.1. Kreisfläche	51
7.2. Quotient	51
7.3. Fakultät	51
7.4. Quadratische Gleichungen (Mitternachtsformel)	52
7.5. Musterlösungen	53
7.5.1. Kreisfläche	53
7.5.2. Quotient	53
7.5.3. Fakultät	54
7.5.4. Mitternachtsformel	55
8. Prüfungsvorbereitung	56
8.1. Belegung von Variablen	56

8.2.	Funktionen	56
8.2.1.	Berechnung des Volumens eines Zylinders	58
8.2.2.	Berechnung der kinetischen Energie eines Körpers	58
8.3.	Musterlösungen	59
8.3.1.	Berechnung des Volumens eines Zylinders	59
8.3.2.	Berechnung der kinetischen Energie eines Körpers	59
9.	Datenstrukturen: Listen	61
9.1.	Daten in Listen speichern	61
9.1.1.	Aufgabe: Monatsliste erstellen	61
9.2.	Index	61
9.2.1.	Aufgabe: Index lesen	62
9.3.	Überschreiben von Elementen	62
9.4.	Schleifen	62
9.4.1.	Aufgabe: Monatsschleife komplettieren	63
9.5.	Länge	63
9.6.	Ergänzen und entfernen	63
9.7.	Aufgabe: Fibonacci	64
9.8.	Aufgabe: zeichne Chomp	64
9.9.	Aufgabe: spiele Chomp	65
9.10.	Übersicht	66
9.11.	Zusatzstoff	66

Willkommen

Hier finden Sie die Informationen für den Informatikunterricht.

Programm

Datum	Thema
18.08.2025	Was ist Informatik
25.08.2025	Vorbereiten der Arbeitsumgebung
01.09.2025	Problemlösung in der Informatik
08.09.2025	Variablen als Wegweiser zu Objekten
15.09.2025	Schlaufen
22.09.2025	Bedingungen
20.10.2025	Anwendungsübung
27.10.2025	Individuelle Prüfungsvorbereitung
03.11.2025	Test
10.11.2025	Datenstrukturen
17.11.2025	Datenstrukturen
24.11.2025	Algorithmen
01.12.2025	Algorithmen
08.12.2025	Wiederholung ohne Schleife (Rekursion)
15.12.2025	Anwendungsübungen
05.01.2026	Individuelle Prüfungsvorbereitung
12.01.2026	Test

Das Programm widerspiegelt den aktuellen Stand der Planung. Es ist im Verlauf des Semesters mit Änderungen zu rechnen.

Materielle Voraussetzungen

Für den Informatikunterricht ist ein Laptop erforderlich (mit einem iPad können Sie die im Unterricht gestellten Aufgaben nicht lösen). Für Ihren Laptop brachen Sie zur Installation der

erforderlichen Software Administratorenrechte.

Ausserdem müssen Sie sicherstellen, dass Ihr Akku zu Beginn des Unterrichts einen Ladestand aufweist, der eine Doppelstunde durchhält.

Beurteilung

Pro Semester sind zwei schriftliche Prüfungen vorgesehen. Ausserdem wird die mündliche Beteiligung benotet. Als mündliche Beteiligung gilt insbesondere auch das Stellen von Fragen.

Die Zeugnisnote berechnet sich als gewichteter Durchschnitt aus den beiden schriftlichen Prüfungen und der Note für die mündliche Beteiligung. Die Durchschnittsnote der beiden schriftlichen Prüfungen wird mit 90%, die Note für die mündliche Beteiligung mit 10% gewichtet.

Falls jemand eine persönliche Besprechung wünscht, kann sich hier für eine Sprechstunde anmelden (Rent a Mock).

Teil I.

Anleitungen

1. Speicherorganisation

Informationen sind in Computern in Dateien gespeichert. Die gespeicherten Informationen können dabei ganz unterschiedlicher Art, wie zum Beispiel Texte, Bilder oder Videos, sein.

Als Modell, wie man sich Dateien vorstellen kann, hat sich das Bild von Dossiers in Ordnern etabliert. Die einzelnen Dossiers sind die Dateien und die Ordner sind die Strukturen, in denen die Dateien abgelegt sind. Diese Konstruktion kann über mehrere Ebenen hinaus verschachtelt werde (Ordner in Gestellen, die wiederum in einzelnen Räumen stehen, etc.).

1.1. Dateinamen und Pfade

Damit Dateien identifiziert und gefunden werden können, müssen sie einen Namen haben. Grundsätzlich gibt es keine Einschränkungen, wie Dateien benannt werden. Die meisten Dateinamen bestehen allerdings aus zwei Teilen: dem eigentlichen Dateinamen und der Dateinamenserweiterung.

Der eigentliche Dateiname wird idealerweise so festgelegt, dass er einen Rückschluss auf den Inhalt der Datei zulässt.

Die Dateinamenserweiterung ist ein Zusatz, der Auskunft über den Dateityp gibt. Sie steht hinter einem Punkt hinter dem eigentlichen Dateinamen. Auf Windows Rechnern wird die Dateinamenserweiterung im Dateimanager in der Standardeinstellung nicht angezeigt. Um dies zu ändern, muss in den Einstellungen des Dateimanagers die Option “Dateinamenserweiterungen anzeigen” aktiviert werden (Ansicht > Anzeigen > Dateinamenserweiterung).

Damit Dateien besser ausgetauscht werden können, empfiehlt es sich, für die Namen lediglich Buchstaben, Zahlen und Unterstriche (sog. [ASCII-Zeichen](#)) zu verwenden.

Damit man Dateien finden kann, muss man wissen, wo sie abgelegt worden sind. Übertragen auf das Modell von Dossiers in Ordnern bedeutet das, zu wissen, welches Dossier in welchem Ordner in welchem Gestell in welchem Raum abgelegt ist. Wie in einem realen Archiv, geht man dabei vom Raum zum Gestell, zum Ordner und schliesslich zum Dossier. In der Informatik wird dieser Weg als Pfad bezeichnet. Auf einem Windows-Rechner beginnt dieser Pfad mit dem sogenannten Laufwerksbuchstaben, gefolgt von einem Doppelpunkt und einem Backslash (\). Aus historischen Gründen ist der Laufwerksbuchstabe auf Windows-Rechnern Standardmässig der Buchstabe C.

Ein Beispiel für einen Pfad könnte so aussehen:


```
1 C:\Users\fritz\Documents\text.docx
```

In diesem Beispiel ist **C:** der Laufwerksbuchstabe, **Users** der Ordner, **fritz** der Unterordner, **Documents** der Unterordner von **fritz** und **text.docx** die Datei, die im Ordner **Documents** abgelegt ist. **Users** ist ein von Windows standardmässig angelegter Ordner, in dem die persönlichen Daten der Benutzer abgelegt werden. Der Ordner **fritz** ist der persönliche Ordner des Benutzers **fritz**. Der Ordner **Documents** wird ebenfalls standardmässig von Windows im Ordner jedes Benutzers angelegt. Dem Benutzer **fritz** steht es frei, diesen Ordner zu verwenden und darin Dateien oder Unterordner anzulegen.

Der Dateiname **text.docx** verweist mit seiner Dateinamenserweiterung **.docx** auf eine Datei, die mit dem Programm Microsoft Word erstellt worden ist.

1.2. Dateien in der Cloud

Dateien können nicht nur lokal auf dem Computer gespeichert werden. Damit von überall und jederzeit auf Dateien zugegriffen werden kann, werden Dateien in der *Cloud* gespeichert. Dabei handelt es sich um Server in Rechenzentren, die über das Internet erreichbar sind. Beispiele für solche Cloud-Dienst sind OneDrive von Microsoft oder Google Drive.

Ablageorte auf OneDrive werden in Windows direkt in der Verzeichnisstruktur des Betriebssystems angezeigt, solche von Google erhalten auf Windows einen eigenen Laufwerksbuchstaben (**G:**).

Damit auf die Dateien in der Cloud zugegriffen werden kann, ist eine Internetverbindung erforderlich.

1.3. Dateistruktur für die Schule

Für den schulischen Bedarf erscheint es sinnvoll eine Dateistruktur nach Fächern anzulegen. Im Ordner **Documents** wird dazu für jedes Fach ein eigener Unterordner angelegt. Innerhalb der jeweiligen Fachordner kann eine weitere Struktur nach Semester oder nach Thema sinnvoll sein. Ein Beispiel für die Ordnerstruktur eines Erstklässlers an der KBW kann so aussehen:

```
1 Documents\  
2     Schule\  
3     |      Deutsch  
4     |      Franz  
5     |      Mathe  
6     |      WR  
7     |      ...
```

```
8      Privat\  
9      |      Rechnungen  
10     |      ...
```

2. Arbeitsumgebung (Arbeiten mit Jupyter Notebooks)

2.1. Ausgangslage

In der Informatik geht es darum, wie Informationsverarbeitung mit Hilfe von Computern automatisiert werden kann.

Die Automatisierung der Informationsverarbeitung erfordert die Verwendung von Programmiersprachen. Im Informatikunterricht wird in erster Linie mit der Programmiersprache Python gearbeitet.

Im folgenden findet sich eine Anleitung für die Installation der für den Unterricht erforderlichen Programme.

2.2. Installation von Python

Dieser Abschnitt führt Sie Schritt für Schritt durch die Installation von Python auf einem Windows-Rechner.

Microsoft Store Falle

Achten Sie beim Herunterladen von Python darauf, dass Sie sich auf der offiziellen Seite von Python (<https://www.python.org>) und **nicht** im Microsoft Store befinden. Wenn Sie versehentlich die Python Version aus dem Microsoft Store installiert haben, kann das bei der Arbeit an den Schulprojekten zu Problemen führen.

Deinstallieren Sie die Microsoft Version von Python und installieren Sie die Version von der offiziellen Website.

1. Laden Sie die neueste Version von Python von der offiziellen Website herunter: [python.org](https://www.python.org).
2. Führen Sie das heruntergeladene Installationsprogramm durch Doppelklick auf die Datei aus. Stellen Sie sicher, dass Sie die Option “Add Python to PATH” aktivieren, bevor Sie auf “Install Now” klicken.

💡 Die 'PATH'-Umgebungsvariable

Stellen Sie sich die PATH-Variable wie ein Adressbuch für die Kommandozeile (Terminal) vor. Wenn Sie einen Befehl wie `python` eingeben, schaut der Computer in diesem Adressbuch nach, wo das entsprechende Programm zu finden ist.

Indem Sie das Häkchen bei "Add Python to PATH" setzen, fügen Sie die Adresse des Python-Interpreters zu diesem Adressbuch hinzu. Ohne diesen Eintrag weiss der Computer nicht, wo er suchen soll, und meldet, dass er den Befehl nicht kennt.

3. Überprüfen Sie die Installation, indem Sie die Eingabeaufforderung öffnen (Terminal → Windows-Taste + R, dann `cmd` eingeben) und den Befehl `python --version` eingeben. Dies sollte die installierte Python-Version anzeigen.

2.3. Hello World

Es hat sich eingebürgert, dass das erste Programm, das ausgeführt wird, ein Programm ist, das den Text "Hello World" auf dem Bildschirm ausgibt. Um dieser Tradition zu folgen, führen Sie die folgenden Schritte aus:

1. Öffnen Sie ein Terminal (Windows-Taste + R, dann `cmd` eingeben).

💡 Das Terminal

Unter dem Begriff "Terminal" versteht man ein Programm, das eine textbasierte Benutzeroberfläche bereitstellt, um mit dem Betriebssystem zu interagieren. In einem Terminal können Sie Befehle eingeben und erhalten die Ausgaben direkt im Fenster.

2. Geben Sie den Befehl `python` ein, um die Python-Shell zu starten. Die Python Shell sollte ungefähr so, wie das folgende Bild aussehen.

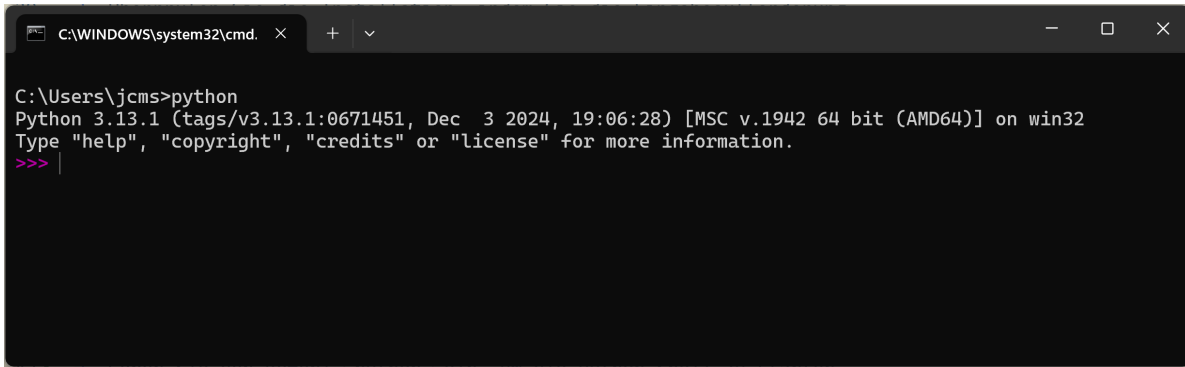


Abbildung 2.1.: Python Shell

3. Geben Sie den folgenden Befehl ein und drücken Sie anschliessend die Eingabetaste:

```
1 print("Hello World")
```

Das Resultat sollte wie das folgende Bild aussehen.

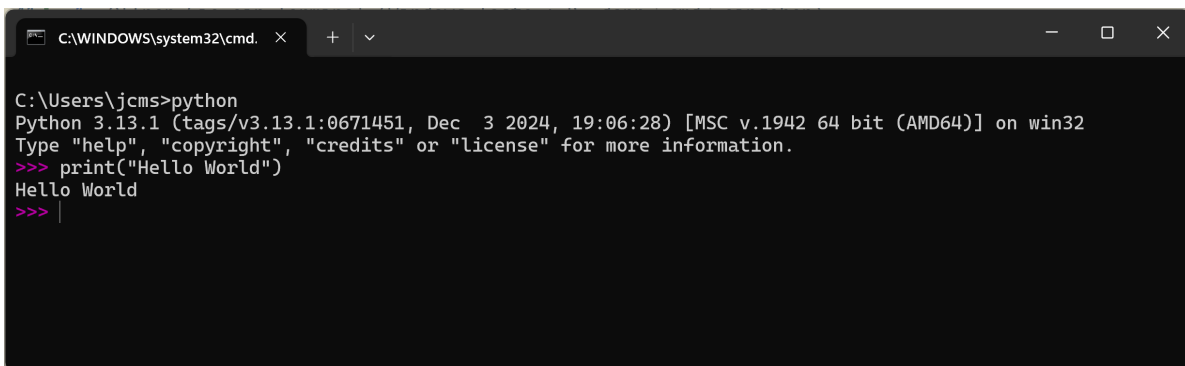


Abbildung 2.2.: Python Shell

Gratuliere - Sie haben Ihr erstes Python-Programm erfolgreich ausgeführt!

2.4. Arbeitsumgebung

Im Unterricht wird nicht direkt in der Python-Shell gearbeitet, sondern mit sogenannten Jupyter Notebooks. Jupyter Notebooks ermöglichen es, in der gleichen Datei sowohl Code (Programm Teile) als auch formatierten Text (in Markdown) zu verarbeiten. Eine Jupyter Notebook Datei hat die Endung `.ipynb`. Vom Jupyter Notebook unterschieden werden muss die Arbeitsoberfläche in welcher die Jupyter Notebooks bearbeitet werden. Diese Oberfläche nennt sich JupyterLab und läuft in einem Webbrowser.

💡 Das Jupyter Ökosystem

Die im Unterricht verwendeten Jupyter Notebooks sind Teil eines ganzen Jupyter Ökosystems. Der Name Jupyter setzt sich aus den drei Programmiersprachen **J**ulia, **P**ython und **R** zusammen, die in diesem Ökosystem eine zentrale Rolle spielen. Zum Jupyter Ökosystem gehören auch zahlreiche Erweiterungen und Tools, die die Arbeit mit Notebooks und Daten erleichtern.

Der Unterricht beschränkt sich auf die Verwendung von Jupyter Notebooks mit der Programmiersprache Python sowie den Einsatz von JupyterLab als Arbeitsumgebung.

Damit dies alles funktioniert, braucht es ein paar weitere Vorbereitungsarbeiten.

1. Erstellen Sie im Ordner “Informatik” einen Unterordner mit dem Heutigen Datum als Namen. Formatieren Sie das Datum nach dem Schema “YYMMDD”, für den 1. August 2025 wäre das zum Beispiel “250801”.
2. Öffnen Sie den soeben erstellten Ordner.
3. Geben Sie die Tastenfolge **Ctrl + L** ein, um die Adresszeile des Dateimanagers zu aktivieren.
4. Überschreiben Sie den Inhalt der Adresszeile mit dem Text `cmd` und drücken Sie die Eingabetaste. Dadurch wird ein Terminal geöffnet, das direkt im aktuellen Ordner arbeitet.
5. Geben Sie im neu geöffneten Terminal den folgenden Befehl ein und drücken Sie die Eingabetaste:

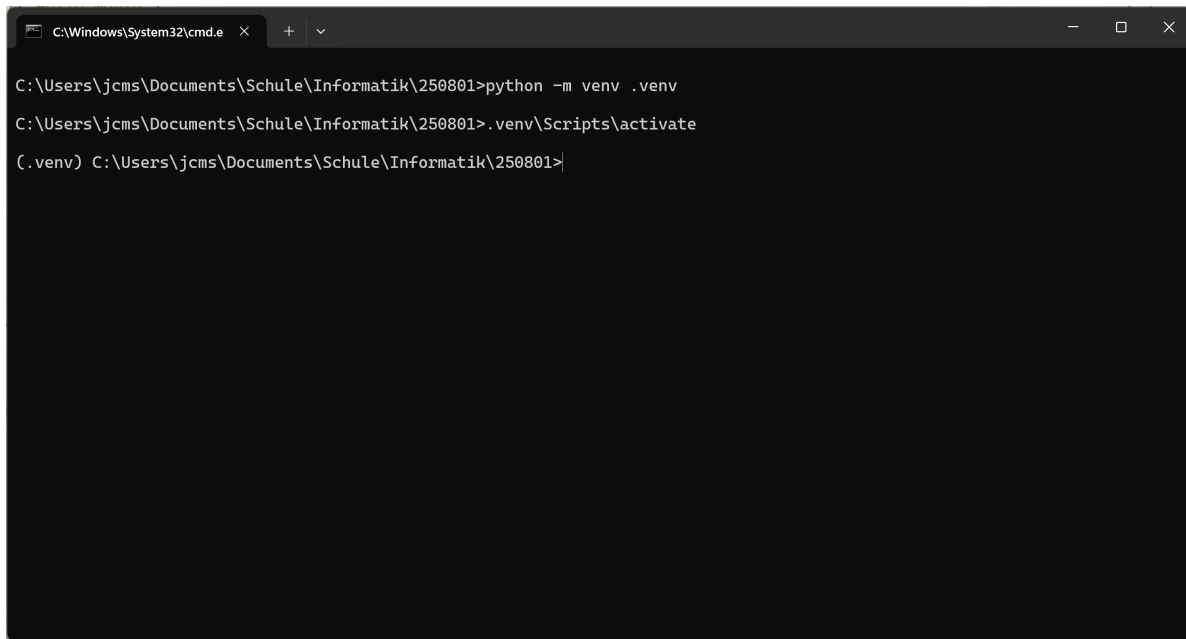
```
1 python -m venv .venv
```

Dadurch wird eine sogenannte Python Virtual Environment erstellt (venv wegen **V**irtual **E**nvironment). Dieses Python Virtual Environment dient dazu, die eigenen Programmierprojekte unabhängig voneinander gestalten zu können.

6. Aktivieren Sie das Python Virtual Environment mit dem folgenden Befehl:

```
1 .venv\Scripts\activate
```

Ihr Terminal sieht nach dem Erstellen und Aktivieren der Python Virtual Environment ungefähr so aus:



```
C:\Windows\System32\cmd.e x + v
C:\Users\jcms\Documents\Schule\Informatik\250801>python -m venv .venv
C:\Users\jcms\Documents\Schule\Informatik\250801>.venv\Scripts\activate
(.venv) C:\Users\jcms\Documents\Schule\Informatik\250801>
```

Abbildung 2.3.: Aktivierte Python Virtual Environment

Das Wort in der Klammer am Anfang der Zeile zeigt den Namen der aktiven Python Virtual Environment an. Im vorliegenden Fall ist das `.venv`.

7. In der nun aktivierten Python Virtual Environment installieren Sie die benötigten Pakete mit dem folgenden Befehl:

```
1 pip install jupyter
```

Das dauert eine Weile.

Während der Installation werden die benötigten Pakete (Ergänzungen zur bestehenden Python Installation) heruntergeladen und in der Python Virtual Environment gespeichert. Das Terminal sieht dabei ungefähr so aus:

```
C:\Windows\System32\cmd.exe x + v
Using cached defusedxml-0.7.1-py2.py3-none-any.whl (25 kB)
Using cached fqdn-1.5.1-py3-none-any.whl (9.1 kB)
Using cached isoduration-20.11.0-py3-none-any.whl (11 kB)
Using cached arrow-1.3.0-py3-none-any.whl (66 kB)
Using cached types_python_dateutil-2.9.0.20250516-py3-none-any.whl (14 kB)
Using cached jupyterlab_pygments-0.3.0-py3-none-any.whl (15 kB)
Using cached nest_asyncio-1.6.0-py3-none-any.whl (5.2 kB)
Downloading notebook-7.4.4-py3-none-any.whl (14.3 MB)
14.3/14.3 MB 2.9 MB/s eta 0:00:00
Using cached psutil-7.0.0-cp37-abi3-win_amd64.whl (244 kB)
Using cached pycparser-2.22-py3-none-any.whl (117 kB)
Using cached rfc3339_validator-0.1.4-py2.py3-none-any.whl (3.5 kB)
Using cached stack_data-0.6.3-py3-none-any.whl (24 kB)
Using cached asttokens-3.0.0-py3-none-any.whl (26 kB)
Using cached executing-2.2.0-py2.py3-none-any.whl (26 kB)
Using cached pure_eval-0.2.3-py3-none-any.whl (11 kB)
Using cached uri_template-1.3.0-py3-none-any.whl (11 kB)
Using cached wcwidth-0.2.13-py2.py3-none-any.whl (34 kB)
Installing collected packages: webencodings, wcwidth, pywin32, pure-eval, fastjsonschema, widgetsnbextension, websocket-client, webcolors, urllib3, uri-template, typing-extensions, types-python-dateutil, traitlets, tornado, tinycss2, soupsieve, sniffio, six, setuptools, send2trash, rpds-py, rfc3986-validator, pyzmq, pyyaml, pywinpty, python-json-logger, pygments, pycparser, psutil, prompt_toolkit, prometheus-client, platformdirs, parso, pandocfilters, packaging, overrides, nest-asyncio, mistune, MarkupSafe, jupyterlab_widgets, jupyterlab_pygments, jsonpointer, json5, idna, h11, fqdn, executing, defusedxml, decorator, debugpy, colorama, charset_normalizer, certifi, bleach, babel, attrs, async-lru, asttokens, terminado, stack_data, rfc3339-validator, requests, referencing, python-dateutil, matplotlib-inline, jupyter-core, Jinja2, jedi, ipython-pygments-lexers, httpcore, comm, cffi, beautifulsoup4, anyio, jupyter-server-terminals, jupyter-client, jsonschema-specifications, ipython, httpx, arrow, argon2-cffi-bindings, jsonschema, isoduration, ipywidgets, ipykernel, argon2-cffi, nbformat, jupyter-console, nbclient, jupyter-events, nbconvert, jupyter-server, notebook-shim, jupyterlab-server, jupyter-lsp, jupyterlab, notebook, jupyter
2/98 [pywin32]
```

Abbildung 2.4.: Terminal während der Jupyter Installation

Alle in einer Python Virtual Environment installierten Pakete sind innerhalb dieser Umgebung dauerhaft verfügbar und müssen daher für das gleiche Projekt kein zweites Mal installiert werden.

8. Starten Sie den Jupyter Server mit dem folgenden Befehl:

```
1 jupyter-lab
```

Dies startet den Jupyter Notebook Server und öffnet automatisch ein Browserfenster mit der Jupyter Notebook Oberfläche.

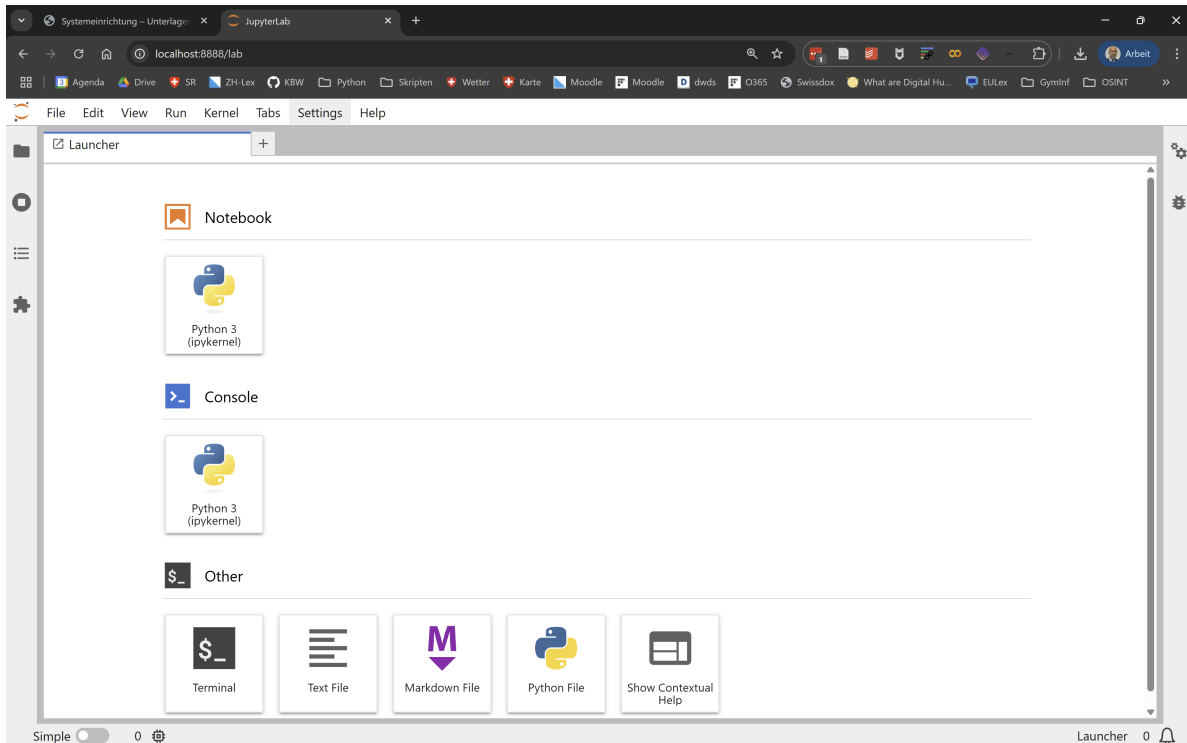


Abbildung 2.5.: Startseite Jupyter Lab

9. Klicken Sie auf den Button “Python 3 (ipykernel)” unter dem Titel Notebook.

Damit starten Sie ein neues Jupyter Notebook. Der Cursor blinkt in einer leeren Zelle. Bei dieser Zelle handelt es sich um eine sogenannte Code-Zelle. In einer Code-Zelle können Sie Python Code eingeben und ausführen.

Überprüfen Sie das, indem Sie in der Zelle den Befehl `print("Hello World")` eingeben und anschliessend die Tastenfolge **Shift + Enter** drücken (alternativ können Sie auch auf den Button “Run” in der Werkzeugleiste klicken).

Das Resultat sollte wie das folgende Bild aussehen.

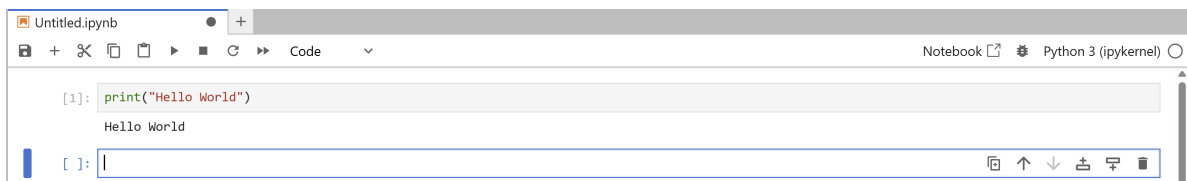


Abbildung 2.6.: Hello World in einem Jupyter Notebook

In einem Jupyter Notebook können Sie nicht nur Python Code ausführen, sondern auch

Text (formatiert in Markdown) darstellen.

Für die Darstellung von Text müssen Sie die Zelle als Text-Zelle markieren. Dazu klicken Sie auf den Button “Code” in der Werkzeugleiste und wählen im Dropdown-Menü die Option “Markdown” aus.

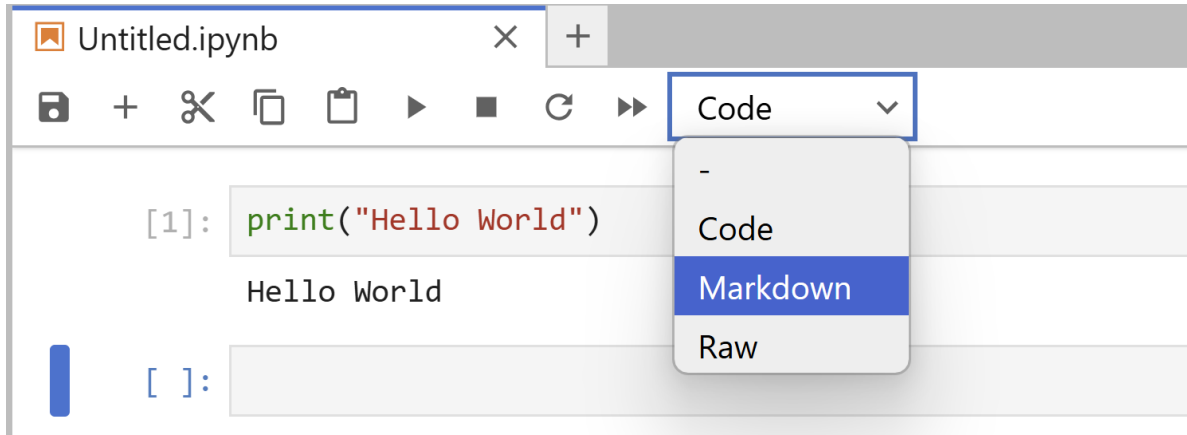


Abbildung 2.7.: Umstellen der Zelle auf Markdown

Probieren Sie das aus. Schreiben Sie einen Titel und einen kurzen Text in die Zelle unterhalb der Code-Zelle mit `print("Hello World")`. Damit der Text in der Zelle formatiert angezeigt wird, müssen Sie die Zelle mit der Tastenfolge **Shift + Enter** ausführen (analog zum Ausführen von Code-Zellen).

Eine Zelle ist entweder eine Code-Zelle oder eine Text-Zelle. Für den Wechsel zwischen Code- und Text-Darstellung müssen Sie je eine neue Zelle anlegen. Das geht mit der Tastenfolge **Esc + B** (für “Below”) oder **Esc + A** (für “Above”) während Sie sich in einer Zelle befinden. Alternativ können Sie auch die Buttons “Insert Cell Below” oder “Insert Cell Above” aus den Werkzeugen der Zelle verwenden.

10. Das Jupyter Notebook ist eine Datei mit der Endung `.ipynb`. Neu erstellte Jupyter Notebooks erhalten den Namen “Untitled.ipynb”. Um diesen Namen zu ändern, klicken Sie mit der rechten Maustaste auf den Titel “Untitled” in der oberen linken Ecke des Jupyter Notebooks und wählen Sie die Option “Rename” aus dem Kontextmenü. Anschliessend können Sie den Namen des Jupyter Notebooks eingeben.

2.5. Öffnen bestehender Jupyter Notebooks

Häufiger als das Erstellen eines neuen Jupyter Notebooks ist das Öffnen eines bereits bestehenden Jupyter Notebooks. Hier wird das entsprechende Vorgehen beschrieben.

1. Navigieren Sie in den Ordner in dem sich das Jupyter Notebook befindet.
2. Stellen Sie sicher, dass der Ordner über eine Python Virtual Environment mit installierten Jupyter Paketen verfügt.

Öffnen Sie dazu im ausgewählten Ordner das Terminal (`ctrl + L` anschliessend `cmd` und Eingabetaste). Dann starten Sie die Python Virtual Environment und geben den Befehl `pip list` ein. Dieser Listet alle in der Python Virtual Environment installierten Pakete auf. Falls die Jupyter Pakete nicht aufgelistet werden, müssen Sie diese wie [oben](#) beschrieben installieren.

3. Starten Sie den Jupyter Server mit dem Befehl `jupyter-lab`.
4. Öffnen Sie das Dateiverzeichnis. Dazu müssen Sie auf dem linken Rand das Ordner-Symbol anklicken.
5. Wählen Sie das Jupyter Notebook aus, das Sie öffnen möchten. Mit einem Doppelklick auf das Jupyter Notebook wird dieses geöffnet.

Wenn Sie das Dateiverzeichnis wieder schliessen möchten, klicken Sie auf das Ordner-Symbol auf der linken Seite erneut.

Diese Schritte funktionieren auch, wenn Sie ein Jupyter Notebook öffnen möchten, das Sie von jemand anderem erhalten haben. Sie müssen dieses dazu lediglich in den Ordner kopieren, in dem sich die Python Virtual Environment mit den installierten Jupyter Paketen befindet.

2.6. Häufige Fehlermeldungen

Problem / Fehlermeldung (Was Sie sehen)	Mögliche Ursache (Warum es passiert)	Lösung (Was Sie tun können)
Der Befehl <code>python</code> ist entweder falsch geschrieben oder konnte nicht gefunden werden.	Python wurde bei der Installation nicht zur PATH-Variable hinzugefügt. Der Computer weiss nicht, wo er <code>python.exe</code> finden soll.	<ol style="list-style-type: none"> 1. Deinstallieren Sie Python über die Systemsteuerung. 2. Installieren Sie Python erneut. 3. Achten Sie diesmal unbedingt darauf, das Häkchen bei "Add Python to PATH" zu setzen.

Problem / Fehlermeldung (Was Sie sehen)	Mögliche Ursache (Warum es passiert)	Lösung (Was Sie tun können)
Der Befehl <code>jupyter-lab</code> ist entweder falsch geschrieben oder konnte nicht gefunden werden.	Sie haben vergessen, die virtuelle Umgebung zu aktivieren. Der Befehl <code>jupyter-lab</code> existiert nur innerhalb der aktivierten Umgebung.	<ol style="list-style-type: none"> 1. Überprüfen Sie, ob <code>(.venv)</code> am Anfang der Kommandozeile steht. 2. Falls nicht, führen Sie den Aktivierungsbefehl erneut aus: <code>.venv\Scripts\activate.</code>
<code>ImportError: DLL load failed...</code> oder ähnliche Fehler unter Windows	Ein häufiges Problem mit der Installation von <code>pywin32</code> , einer wichtigen Windows-Bibliothek, die von Jupyter benötigt wird.	<ol style="list-style-type: none"> 1. Stellen Sie sicher, dass Ihre virtuelle Umgebung aktiv ist. 2. Führen Sie den Befehl <code>pip install --upgrade pywin32</code> aus, um die Bibliothek zu reparieren.
Kernel Error oder der Status "Kernel starting, please wait..." ändert sich nicht	Die Verbindung zwischen der Browser-Oberfläche und dem Python-"Gehirn" (dem Kernel) ist gestört. Dies kann viele Ursachen haben.	<ol style="list-style-type: none"> 1. Der einfachste erste Schritt: Klicken Sie im JupyterLab-Menü auf "Kernel" -> "Restart Kernel..." 2. Wenn das nicht hilft, schliessen Sie JupyterLab im Terminal (mit der Tastenkombination <code>Strg + C</code>) und starten Sie es mit <code>jupyter-lab</code> neu.

`Permission denied` (Zugriff
verweigert) bei der
Installation von Paketen

Sie versuchen, Pakete an
einem systemweiten Ort zu
installieren (z. B. in
`C:\Program Files`), für den
Sie keine Schreibrechte
haben.

Dies ist genau das Problem,
das virtuelle Umgebungen
lösen! Stellen Sie sicher, dass
Ihre `venv` aktiv ist (`.venv`
muss sichtbar sein). Dadurch
wird sichergestellt, dass alle
Pakete lokal in Ihren
Projektordner installiert
werden, wo Sie die vollen
Rechte haben.

Teil II.

Einführung in Python

3. Problemlösung in der Informatik

3.1. Ausgangslage

Jede Disziplin hat ihre eigene Art, Probleme zu lösen. Das ist in der Informatik nicht anders. In der Informatik versucht man, grosse Probleme in kleinere Teilprobleme zu zerlegen. Das macht man so lange, bis die Teilprobleme so klein sind, dass sie einfach zu lösen sind.

Dies soll hier anhand von verschiedenen Grafiken gezeigt werden.

Damit in Python einfach mit Grafiken gearbeitet werden kann, wird das Paket **PyTamaro** verwendet. Dieses Paket wurde von der Università della Svizzera italiana (USI) extra für die Informatik-Ausbildung entwickelt.

Damit das Paket verwendet werden kann, muss es zuerst in der aktuellen Python Virtual Environment installiert werden.

Dazu öffnen Sie ein Terminal im Ordner, in dem sich dieses Jupyter Notebook befindet. Anschliessend starten Sie die Python Virtual Environment mit dem Befehl:

```
1 .venv\Scripts\activate
```

Danach können Sie **PyTamaro** mit dem Befehl

```
1 pip install pytamaro
```

installieren.

Um das Paket zu verwenden, muss es *importiert* werden. Die genauen Zusammenhänge müssen im Moment nicht bekannt sein. Wichtig ist lediglich, dass die folgende Zelle ausgeführt wird.

```
1 from pytamaro.de import (  
2     rechteck, kreis_sektor,  
3     blau, rot, weiss, schwarz,  
4     neben, ueber, ueberlagere,  
5     drehe, kombiniere,  
6     zeige_grafik, speichere_grafik,  
7 )
```

3.2. Beispiel: Tricolore

Die Vorgehensweise wird anhand der Französischen Nationalflagge (Tricolore) gezeigt.

Um die Zeichnung der Tricolore zu planen, wird die Grafik in ihre Einzelteile zerlegt. Die Tricolore besteht aus drei gleich grossen Rechtecken in den Farben blau, rot und weiss. Diese Rechtecke werden nebeneinander angeordnet.

Das bedeutet, dass die Länge und die Breite der Rechtecke definiert werden muss und basierend auf diesen Werten die drei Rechtecke gezeichnet werden. Anschliessend werden die drei Rechtecke nebeneinander angeordnet.

Der Befehl zum Zeichnen eines Rechtecks lautet

```
1 name = rechteck(länge, breite, farbe)
```

Bevor die Zeichnung tatsächlich erstellt wird, soll hier der Befehl im Detail erklärt werden:

- **name** ist der Name, unter dem das Rechteck gespeichert wird. Dieser Name kann später verwendet werden, um auf das Rechteck zuzugreifen.
- **rechteck** ist der Befehl, der ein Rechteck zeichnet. In der Klammer hinter dem Befehl werden die sogenannten *Argumente* angegeben. Diese steuern, wie das Rechteck aussieht.
- **länge** und **breite** sind die Argumente, die die Grösse des Rechtecks bestimmen. Diese Werte können beliebig gewählt werden.
- **farbe** ist das Argument, das die Farbe des Rechtecks bestimmt. Aufgrund der Eigenheiten von PyTamaro können ausschliesslich die Farben verwendet werden, welche importiert worden sind.

3.2.1. Rechtecke zeichnen

Als erstes wird hier gezeigt, wie das blaue Rechteck gezeichnet wird.

Damit das Resultat kontrolliert werden kann, wird die Grafik mit dem Befehl

```
1 zeige_grafik(name)
```

angezeigt.

```
1 bleu = rechteck(50, 100, blau)
2 zeige_grafik(bleu)
```



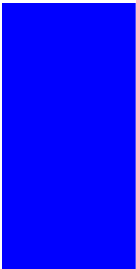

Nachdem das blaue Rechteck gezeichnet wurde, kann das weisse und das rote Rechteck analog gezeichnet und angezeigt werden.

```
1 # TODO: Rechtecke blanc und rouge zeichnen -> Schreiben Sie hier Ihren
2 # Code
3 blanc = rechteck(50, 100, weiss)
4 rouge = rechteck(50, 100, rot)
```

Als nächstes werden die drei Rechtecke nebeneinander angeordnet. Dazu wird der Befehl **neben** verwendet. Dieser Befehl nimmt zwei Argumente entgegen: das erste Rechteck und das zweite Rechteck. Das erste Rechteck wird links vom zweiten Rechteck gezeichnet.

```
1 resultat = neben(linker grafik, rechter grafik)
```

```
1 zwei_drittel = neben(bleu, blanc)
2 zeige_grafik(zwei_drittel)
```



Analog können Sie nun das rote Rechteck rechts der zwei Drittel anordnen. Nennen Sie das Resultat **tricolore** und zeigen Sie es an.

```
1 # TODO: Tricolore zusammenfügen -> Schreiben Sie hier Ihren
2 # Code
```

3.3. Beispiel: Österreichische Flagge

Zeichnen Sie die Österreichische Flagge. Das Seitenverhältnis der Flagge ist 2:3.

Um Elemente übereinander anzuordnen, wird der Befehl `ueber` verwendet. Die Syntax dieses Befehls lautet:

```
1 resultat = ueber(obere grafik, untere grafik)
```

```
1 # TODO: Östereichische Flagge zeichnen -> Schreiben Sie hier Ihren  
2 # Code
```

3.4. Beispiel: Schweizerfahne

Zeichnen Sie eine korrekt propotionierte Schweizerfahne. Die Dimensionen können Sie der folgenden Grafik entnehmen:

Verwenden Sie dazu die Befehle `rechteck`, `drehe` und `ueberlagere`.

```
1 # TODO: Schweizerfahne zeichnen -> Schreiben Sie hier Ihren  
2 # Code
```

3.5. Beispiel: Tessiner Wappen

Als Referenz an die USI zeichnen Sie als letztes Beispiel das Tessiner Wappen.

Verwenden Sie dazu neben den bereits bekannten Befehlen zusätzlich die Befehle `ueber` und `kreis_sektor`.

```
1 # TODO: Tessiner Wappen zeichnen -> Schreiben Sie hier Ihren  
2 # Code
```

3.6. Musterlösungen

3.6.1. Trikolore

```

1  bleu = rechteck(50, 100, blau)
2  blanc = rechteck(50, 100, weiss)
3  rouge = rechteck(50, 100, rot)
4
5  deux_tiers = neben(bleu, blanc)
6  tricolore = neben(deux_tiers, rouge)
7
8  zeige_grafik(tricolore)

```

3.6.2. Österreichische Flagge

```

1  roter_balken = rechteck(90, 20, rot)
2  weisser_balken = rechteck(90, 20, weiss)
3
4  oberer_teil = ueber(roter_balken, weisser_balken)
5  oesterreichische_flagge = ueber(oberer_teil, roter_balken)
6
7  zeige_grafik(oesterreichische_flagge)

```

3.6.3. Schweizerfahne

```

1  hintergrund = rechteck(320, 320, rot)
2  weisser_balken = rechteck(200, 60, weiss)
3  weisser_balken2 = drehe(90, weisser_balken)
4
5  kreuz = ueberlagere(weisser_balken, weisser_balken2)
6
7  schweizerfahne = ueberlagere(kreuz, hintergrund)
8
9  zeige_grafik(schweizerfahne)

```

3.6.4. Tessiner Wappen

```

1  roter_teil = rechteck(160, 240, rot)
2  blauer_teil = rechteck(160, 240, blau)
3  roter_sektor = kreis_sektor(160, 90, rot)

```

```
4  roter_sektor = drehe(180, roter_sektor)
5  blauer_sektor = kreis_sektor(160, 90, blau)
6  blauer_sektor = drehe(270, blauer_sektor)
7
8  ti_unten = neben(roter_sektor, blauer_sektor)
9  ti_oben = neben(roter_teil, blauer_teil)
10
11 tessin = ueber(ti_oben, ti_unten)
12
13 zeige_grafik(tessin)
```

4. Variablen in Python

4.1. Vorbemerkungen: Python als Rechner

Python verfügt über eingebaute mathematische Fähigkeiten. Es kann die Grundrechenarten und kennt die Hierarchie der Operationen. Sie können das überprüfen, in dem Sie in der folgenden Zelle die Rechnung

$$2 + 3 \cdot 4$$

ausführen.

```
1 # hier können Sie die Rechnung ausführen
```

Die folgende Tabelle gibt einen Überblick über die direkt in Python verfügbaren mathematischen Funktionen:

Beschreibung	Befehl	Beispiel	Resultat
Addition	+	$2 + 3$	5
Subtraktion	-	$3 - 2$	1
Multiplikation	*	$3 * 2$	6
Division	/	$3 / 2$	1.5
Potenzen	**	$3 ** 2$	9
Wurzeln	$**(1/n)$	$16 ** (1/2)$	4.0
Ganzzahlige Division	//	$7 // 2$	3
Modulo	%	$7 \% 2$	1

4.2. Variablen

In Python sind Variablen symbolische Namen für gespeicherte Daten. Variablen verweisen dabei auf den Speicherbereich im Computer, in welchem die entsprechenden Daten physikalisch abgelegt sind. Aus diesem Grund werden Variablen gelegentlich auch als Zeiger bezeichnet. Was genau für Daten in diesem Speicherbereich abgelegt werden, spielt keine Rolle und kann während der Ausführung eines Programmes auch ändern.

In Python werden Variablen Werte mit dem Gleichheitszeichen zugewiesen. Um der Variable x den Wert 2 zuzuweisen, ist die Eingabe `x = 2` erforderlich. Die Variable muss links vom Gleichheitszeichen, der zuzuweisende Wert rechts davon stehen.

Überprüfen Sie dies, indem Sie in der folgenden Zelle der Variabel y den Wert 3 und der Variabel z den Wert 4 zuweisen. Anschliessend multiplizieren Sie die beiden Variablen miteinander.

```
1 # hier die Aufgabe einfüllen
```

Wenn Variablen neue Werte zugewiesen werden, wird die Referenz auf den Speicherbereich mit dem alten Wert gelöscht. Die Daten, welche ohne Verweis durch eine Variable im Speicher liegen, werden vom in Python eingebauten *Garbage Collector* im Hintergrund gelöscht und der so freigewordene Speicherplatz kann wieder verwendet werden.

Sie können überprüfen, dass Variablen neue Werte zugewiesen werden können, indem Sie in der untenstehenden Zelle die Variablen y und z addieren. Sie erhalten dann das Resultat 7. Das heisst, den Variablen y und z sind immer noch die Werte 3 und 4 zugewiesen.

```
1 # addieren Sie hier y und z
```

Wenn Sie in der folgenden Zelle der Variabel y den Wert 5 zuweisen und anschliessend y und z addieren erhalten Sie als neues Resultat 9.

```
1 # weisen Sie hier y den neuen Wert zu
```

Variablen können auch Resultate von Berechnungen zugewiesen werden. Ausserdem können Variablen ganze Wörter als Namen haben. Dies ist gegenüber einzelnen Buchstaben vorzuziehen, weil dann aussagekräftige Namen gewählt werden können. Grundsätzlich sind die Namen von Variablen frei wählbar. Es gibt allerdings eine Reihe von [reservierten Begriffen](#), welche in der Programmiersprache Python eine eigene Bedeutung haben. Unzulässig sind ausserdem Namen, die mit Ziffern beginnen.

Für die Darstellung von Namen für Variablen hat sich in Python eingebürgert, Variablen klein zu schreiben und Wörter durch Underlines zu trennen (*this_is_a_valid_variable*). Diese Darstellung nennt sich *Snake Case*. Zudem werden Variablen meist mit englischen Begriffen bezeichnet.

Weisen Sie in der nächsten Zelle der Variable `result` das Resultat der Rechnung $y + z$ zu und geben Sie das Resultat mit `print(result)` aus. `print()` ist eine Funktion, die Python zur Verfügung stellt. Was Funktionen sind, wird im nächsten Abschnitt erklärt.

```
1 # weisen Sie hier der Variable result das Resultat zu
```

4.3. Funktionen in Python

Python verfügt über viele bereits vordefinierte Funktionen. Die oben verwendete Funktion `print()` ist ein Beispiel dafür. Um zu demonstrieren, wie Funktionen in Python definiert werden, zeige ich Ihnen als Beispiel eine Funktion, mit der zwei Zahlen addiert werden.

```
1 # Definition der Funktion
2 def get_sum(x, y):
3     return x + y
4
5 # Aufruf der Funktion
6 result = get_sum(3,4)
7 # Ausgabe des Resultats des Funktionsaufrufs
8 print(result)
```

`def` ist das Schlüsselwort für die Definition einer Funktion. `get_sum` ist der von mir gewählte Name dieser Funktion. Für die Wahl des Namens einer Funktion gelten die gleichen Regeln, wie für Variablen. In den Klammern stehen die sogenannten Parameter, welche der Funktion übergeben werden, damit sie etwas damit macht. Mit dem Doppelpunkt wird die *Signatur* der Funktion abgeschlossen. Die *Signatur* zeigt idealerweise, *was* eine Funktion *womit* macht. Sie gibt aber keine Auskunft darüber, wie sie das macht.

Python gruppiert Befehle, die zusammengehören, durch die gleiche Tiefe der Einrückung. Eine Einrückung hat üblicherweise die Tiefe von vier Leerzeichen. Im Beispiel oben gibt es nur eine eingerückte Zeile, weil die Funktion nur aus einem Befehl besteht. Mit `return` gibt die Funktion das Resultat zurück.

Im Beispiel wird das Resultat der Berechnung, welche die Funktion ausführt der Variable `result` zugewiesen. Der Wert der Variable `result` wird mit `print(result)` ausgegeben.

Definieren Sie in der folgenden Zelle eine Funktion, mit der zwei Zahlen multipliziert werden.

```
1 # hier kommt Ihre Funktion hin
```

4.4. Datentypen

Als nächstes geht es um die Frage, auf welche Inhalte eine Variable zeigen kann.

Im Grundsatz kann eine Variable auf beliebige Inhalte verweisen.

Am einfachsten ist die Verwendung der grundlegenden Datentypen (basic data types), welche Python zur Verfügung stellt. Dies sind (mit ihren englischen Bezeichnungen):

- Integer (Ganzzahl)
- Floating-Point Number (Gleitkommazahl)
- Complex Number (komplexe Zahl)
- String (Zeichenkette)
- Boolean Type (Wahrheitswert)

Darüber hinaus ist es möglich, eigene Datentypen zu programmieren. Hier aber zuerst eine Beschreibung der grundlegenden Datentypen von Python.

4.4.1. Integer

Die Bezeichnung für Integer in Python ist ein kurzes `int`.

Anders als in anderen Programmiersprachen gibt es in Python theoretisch keine Beschränkung, wie gross ein Integer sein kann. Die einzige Grenze ist der Speicherplatz des auf dem der Integer gespeichert werden soll.

Wenn einer Variable ein grosser Integer zugewiesen wird, kann dieser zur besseren Lesbarkeit auch mit einem Underline als Tausendertrennzeichen geschrieben werden (`100_000`).

Um das Auszuprobieren, weisen Sie in der folgenden Zelle der Variable *a* den Wert von einer Million und der Variable *b* den Wert von einer Milliarde zu. Anschliessend addieren Sie *a* und *b* und weisen das Resultat der Variable *big_sum* zu. Zum Schluss geben Sie den Wert von *big_sum* mit der Funktion `print()` aus.

```
1 # hier können Sie Ihre Berechnung vornehmen
```

Eingegebene Zahlen werden automatisch als Dezimalzahlen interpretiert.

Integers können jedoch auch als Binär-, Oktal- oder Hexadezimalzahlen eingegeben werden. Die Eingabe erfordert dann allerdings ein Präfix, welches das Zahlensystem identifiziert. Die folgende Tabelle stellt die möglichen Präfixe zusammen.

Präfix	Bedeutung	Basis
0b (Null + Kleinbuchstabe b)	Binärzahl	2
0B (Null + Grossbuchstabe B)		2
0o (Null + Kleinbuchstabe o)	Oktalzahl	8
0O (Null + Grossbuchstabe O)		8
0x (Null + Kleinbuchstabe x)	Hexadezimalzahl	16
0X (Null + Grossbuchstabe X)		16

In der folgende Zelle finden Sie ein entsprechendes Beispiel.

Python kann Integer in verschiedenen Zahlensystemen darstellen. Um das Zahlensystem bei der Zuweisung zu spezifizieren, verwenden Sie die entsprechenden Präfixe. Die folgenden Beispiele zeigen die Zuweisung der Zahlen 10, 8, 255 und 16 in den Zahlensystemen Dezimal, Oktal, Hexadezimal und Binär:

```
1 dezimal      = 10
2 oktal       = 0o10
3 hexadezimal = 0x10
4 binaer      = 0b10
```

Die Anzeige der Werte der entsprechenden Variablen erfolgt grundsätzlich im Dezimalsystem.

```
1 b = 0b101010
2 o = 0o52
3 x = 0x2a
4
5 print(b, o, x)
```

4.4.2. Gleitkommazahl

Die Bezeichnung für Gleitkommazahlen in Python ist `float`. Python interpretiert Zahlen mit einem Dezimalpunkt als Gleitkommazahlen. Optional können Zahlen mit `e` oder `E` in “wissenschaftlicher” Schreibweise eingegeben werden ($1000 = 1e3$ bzw. $1e-3 = 0.001$).

Weisen Sie in der folgenden Zelle den Variablen `million` und `billionth` die passenden Werte in wissenschaftlicher Schreibweise zu.

```
1 # hier die Werte den beiden Variablen zuweisen
```

4.4.3. Komplexe Zahlen

Python kann auch mit komplexen Zahlen umgehen. Der Abschnitt zu diesem Thema kann wieder aufgegriffen werden, wenn Sie in Mathe die komplexen Zahlen besprochen haben.

4.4.4. String

Zeichenketten (String) werden von Python als `str` bezeichnet.

Zeichenketten sind beliebige Zeichenfolgen. Damit Python Zeichenketten als solche erkennt, müssen sie durch die Verwendung von einfachen oder doppelten Anführungs- und Schlusszeichen als solche gekennzeichnet werden.

"Ich bin eine Zeichenkette." oder 'Ich bin auch eine Zeichenkette.'

Wenn man innerhalb einer Zeichenkette Anführungszeichen braucht, müssen die eingrenzenden Anführungszeichen von der "anderen Sorte" sein ("It's cool learning Python!" oder 'Der Lehrer sagt: "Es ist cool Python zu lernen."'). Eine andere Möglichkeit reservierte Zeichen zu verwenden ist der Gebrauch eines "escape"-Zeichens. In Python ist das der "backslash" (`\`). Die beiden Beispielsätze von vorher hätten entsprechend auch folgendermassen geschrieben werden können:

'It\'s cool learning Python!' bzw. "Der Lehrer sagt: \"Es ist cool Python zu lernen.\\\""

Die Länge von Zeichenketten wird lediglich durch die Speicherkapazität des verwendeten Systems begrenzt. Zeichenketten können nicht nur sehr lang, sondern auch leer sein (`''`).

Zeichenketten können, wie alle Datentypen, Variablen zugewiesen werden. Dies zeigt das folgende Beispiel.

```
1 # Zuweisung eines Strings zu einer Variabel
2 standard_greeting = "Hello World"
3
4 # Ausgabe der Variabel
5 print(standard_greeting)
```

4.4.5. Boolean Type

Wahrheitswerte werden in Python als `bool` bezeichnet. Wahrheitswerte können entweder "wahr" oder "falsch" sein.

```
1 wahr      = True
2 falsch    = False
```

Die Ausgabe findet sich in der folgenden Zelle.

```
1 wahr      = True
2 falsch    = False
3 print(wahr, falsch)
```

Welche Wahrheitswerte sich bei Vergleichen ergeben, kann mit den folgenden Operatoren überprüft werden:

```
1 == # Gleichheit
2 != # Ungleichheit
3 > # Größer als
4 < # Kleiner als
5 >= # Größer oder gleich
6 <= # Kleiner oder gleich
```

```
1 # Testen Sie hier den Wahrheitswert von Vergleichen mit Zahlen
```

Wahrheitswerte werden zur Steuerung von Programmflüssen verwendet. Mit einem Wahrheitswert kann zum Beispiel gesteuert werden, wie oft ein Programmteil wiederholt werden soll.

4.5. Funktionen mit Type-Hints

Zum Abschluss komme ich noch einmal auf die Definition von Funktionen zurück. In Python können Variablen - anders als zum Beispiel in Java - beliebige Datentypen zugewiesen werden. Wenn Variablen im Verlauf eines Programms mehrfach verwendet werden, können ihnen auch unterschiedliche Datentypen zugewiesen werden. Dies ist allerdings schlechter Programmierstil.

Aus diesem Grund ist es sinnvoll, bei der Definition einer Funktion zu deklarieren, welche Datentypen die Parameter haben und welcher Datentyp der Rückgabewert hat. Dies soll mit dem Beispiel der Funktion `get_quotient` verdeutlicht werden.

```
1 def get_quotient(x : int, y : int) -> float:
2     return x / y
```

Hier wird angegeben, dass die Parameter x und y vom Datentyp `int` sein sollen. Der Datentyp des Rückgabewertes wird hinter `->` geschrieben. Im Beispiel ist der Rückgabewert vom Typ `float`. Das ist so, weil die Funktion zum Beispiel $3/4 = 3.5$ rechnet.

Aber Achtung: die Funktion arbeitet auch dann korrekt, wenn ein anderer als der deklarierte Datentyp übergeben wird. Voraussetzung ist lediglich, dass der Datentyp mit den verwendeten Operationen kompatibel ist. Die “Type-Hints” dienen lediglich der besseren Nachvollziehbarkeit, was die Funktion macht.

4.6. Ausgewählte Musterlösungen

4.6.1. Funktion zur Multiplikation zweier Zahlen

```
1 def get_product(a, b):  
2     return a * b  
3  
4 product = get_product(3, 4)  
5 print(product)
```

4.6.2. Gleitkommazahlen in wissenschaftlicher Schreibweise

```
1 million = 1e6  
2 billonth = 1e-9  
3 print(million, billonth)
```

1000000.0 1e-09

5. Wiederholungen in Python (For-Loops)

Eine Stärke von Computerprogrammen ist die wiederholte Ausführung von Anweisungen. Viele Programmiersprachen stellen dafür ein Konstrukt mit dem Namen ‘For-Schleife’ zur Verfügung. Eine ‘For-Schleife’ funktioniert unabhängig von einer konkreten Programmiersprache folgendermassen:

```
FÜR variable VON startwert BIS endwert [MIT schrittweite]
    Anweisungen
ENDE FÜR
```

Übersetzt nach Python sieht das so aus:

```
1 for i in range(n):
2     do...
```

`startwert BIS endwert [MIT schrittweite]` wird dabei durch `range(n)` ausgedrückt. Dabei ist `n` der Endwert. Gezählt wird bis zum aber ohne den Endwert. Startwert und Schrittweite haben Vorgabewerte. Der Vorgabewert für den Start ist 0, derjenige der Schrittweite 1. Weil `range()` diese vorgegebenen Werte hat, müssen diese nicht explizit angegeben werden. Wenn der Startwert abweichend vom Vorgabewert festgelegt werden soll, kann dieser explizit angegeben werden. Der Aufruf von `range()` sieht dann so aus:

```
1 range(startwert, endwert)
```

Falls eine von 1 abweichende Schrittweite festgelegt werden soll lautet der Aufruf

```
1 range(startwert, endwert, schrittweite)
```

In diesem Fall müssen neben dem Endwert sowohl der Startwert und die Schrittweite angegeben werden. Andernfalls kann nicht zwischen den einzelnen Angaben zu Endwert, Startwert und Schrittweite unterschieden werden.

5.1. Anwendungsübung zu Wiederholungen in Python

Im Folgenden bauen Sie eine Blume mit Python for-loops und PyTamaro.

In der folgenden Zelle werden zuerst [alle von PyTamaro zur Verfügung gestellten Funktionen](#) importiert.

```
1 from pytamaro.de import *
```

Das Ziel ist es, eine Blume wie die abgebildete zu zeichnen.

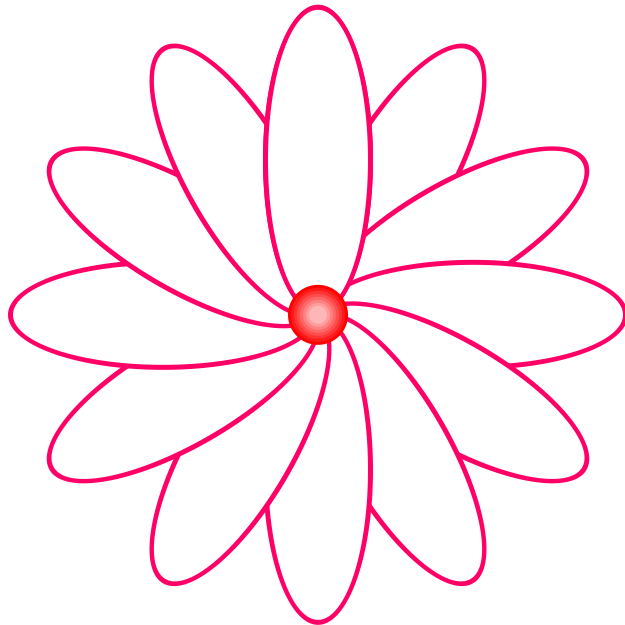


Abbildung 5.1.: Blume

Die wichtigsten Funktionen von PyTamaro für diese Aufgabe sind `fixiere()` und `kombiniere()`.

Die Funktion `fixiere()` legt für eine Grafik einen frei gewählten Ankerpunkt fest. Dazu muss man wissen, dass jede in PyTamaro gezeichnete Grafik von einer sogenannten “Bounding Box” umgeben ist. Diese Bounding Box ist ein Rechteck, das die Grafik vollständig umschliesst. Standardmässig liegt der Ankerpunkt einer Grafik in der Mitte der Bounding Box. Mit `fixiere()` kann man den Ankerpunkt jedoch an eine andere Stelle der Bounding Box verschieben. PyTamaro stellt die folgenden Positionen für Ankerpunkte zur Verfügung:

- `mitte`: Mitte der Bounding Box (Standard)
- `mitte_links`: Mitte der linken Seite der Bounding Box

- `mitte_rechts`: Mitte der rechten Seite der Bounding Box
- `oben_links`: Obere linke Ecke der Bounding Box
- `oben_mitte`: Mitte der oberen Seite der Bounding Box
- `oben_rechts`: Obere rechte Ecke der Bounding Box
- `unten_links`: Untere linke Ecke der Bounding Box
- `unten_mitte`: Mitte der unteren Seite der Bounding Box
- `unten_rechts`: Obere rechte Ecke der Bounding Box

Der Ankerpunkt wird mit dem Befehl `fixiere(position, grafik)` gesetzt.

Der Ankerpunkt ist wichtig für die Funktion `drehe()`, die eine Grafik um einen bestimmten Winkel dreht. Die Drehung erfolgt immer um den Ankerpunkt.

Die Funktion `kombiniere(vordere Grafik, hintere Grafik)` fügt zwei Grafiken zusammen. Die erste gegebene Grafik liegt im Vordergrund und die zweite im Hintergrund. Die Grafiken werden so ausgerichtet, dass ihre Ankerpunkte übereinanderliegen.

5.1.1. Schritt 1: Zerlege das Bild in seine Einzelteile

Die Blume besteht im wesentlichen aus zwei Formen:

1. Blütenblätter in Form von Ellipsen sowie
2. einer Scheibe in der Form eines Kreises in der Mitte.

Um die Blume zu zeichnen, müssen diese beiden Formen zuerst einzeln erstellt werden.

```

1 bluetenblatt = ellipse(50, 150, blau)
2 # die Farbe blau wurde gewählt, damit das Blatt vor dem Hintergrund
3 # sichtbar ist
4
5 zeige_grafik(bluetenblatt)
```



```
1 scheibe = ellipse(30, 30, rot)
2 zeige_grafik(scheibe)
```



5.1.2. Schritt 2: Positioniere die Blütenblätter

Die Blütenblätter sind rund um die Mitte der Blume angeordnet. Da es zwölf Blütenblätter sind, beträgt der Winkel zwischen zwei Blütenblättern $\frac{360^\circ}{12} = 30^\circ$. Die Drehung der Blütenblätter erfolgt um den Ankerpunkt `unten_mitte` der Blütenblätter.

Die Blütenblätter müssen also zuerst fixiert und dann gedreht werden.

```
1 bluetenblatt_fixiert = fixiere(unten_mitte, bluetenblatt)
2 bluetenblatt_30 = drehe(30, bluetenblatt_fixiert)
3 zeige_grafik(bluetenblatt_30)
```



5.1.3. Schritt 3: Zeichnen aller erforderlichen Blütenblätter

Die Blume hat zwölf Blütenblätter. Diese können von Hand jedes einzelne erstellt werden. Das dritte Blütenblatt ist um 60° gedreht und wird entsprechend mit

```
1 bluetenblatt_60 = drehe(60, bluetenblatt_fixiert)
```

erstellt. Das kann man für alle zwölf Blütenblätter machen bis man beim zwölften Blütenblatt angekommen ist, das um 330° gedreht ist.


```
1 blutenblatt_330 = drehe(330, blutenblatt_fixiert)
```

Das ist aber sehr mühsam und fehleranfällig. Viel einfacher ist es, eine Schleife (**for-loop**) zu verwenden, die die Blütenblätter basierend auf der Grundform automatisch erstellt.

```
1 for i in range(12):  
2     winkel = i * 30  
3     blutenblatt_gedreht = drehe(winkel, blutenblatt_fixiert)  
4  
5 zeige_grafik(blutenblatt_gedreht)
```



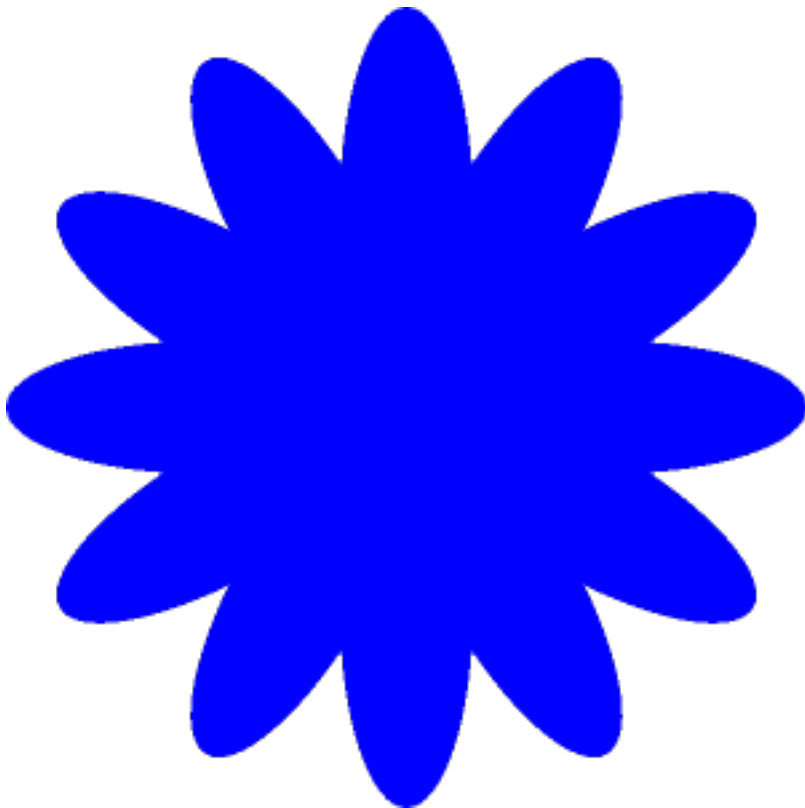
Diese Schleife durchläuft die Zahlen von 0 bis 11 (also 12 Zahlen) und berechnet für jede Zahl den entsprechenden Drehwinkel. Dann wird das Blütenblatt um diesen Winkel gedreht. Allerdings wird das gedrehte Blütenblatt in jeder Iteration der Schleife in der gleichen Variable `blutenblatt_gedreht` gespeichert. Am Ende der Schleife enthält diese Variable nur das letzte gedrehte Blütenblatt (das um 330° gedrehte Blütenblatt).

5.1.4. Schritt 4: Kombiniere alle Blütenblätter

Um das unerwünschte Verhalten zu vermeiden, dass am Ende der Schleife nur das letzte gedrehte Blütenblatt in der Variable `blutenblatt_gedreht` enthalten ist, müssen die gedrehten Blütenblätter in jeder Iteration der gewünschten Gesamtgrafik hinzugefügt werden. Dazu wird die Funktion `kombiniere()` verwendet.

```
1 blume = blutenblatt_fixiert  
2  
3 for i in range(12):  
4     winkel = i * 30  
5     blutenblatt_gedreht = drehe(winkel, blutenblatt_fixiert)  
6     blume = kombiniere(blume, blutenblatt_gedreht)
```

```
7
8 zeige_grafik(blume)
```



Damit sind die Blütenblätter in der gewünschten Position und Anzahl zusammengefügt. Damit Sie allerdings aussehen, wie die Blume in der Vorlage, müssen die einzelnen Blütenblätter weiss eingefärbt und mit einem rosa Rand versehen werden.

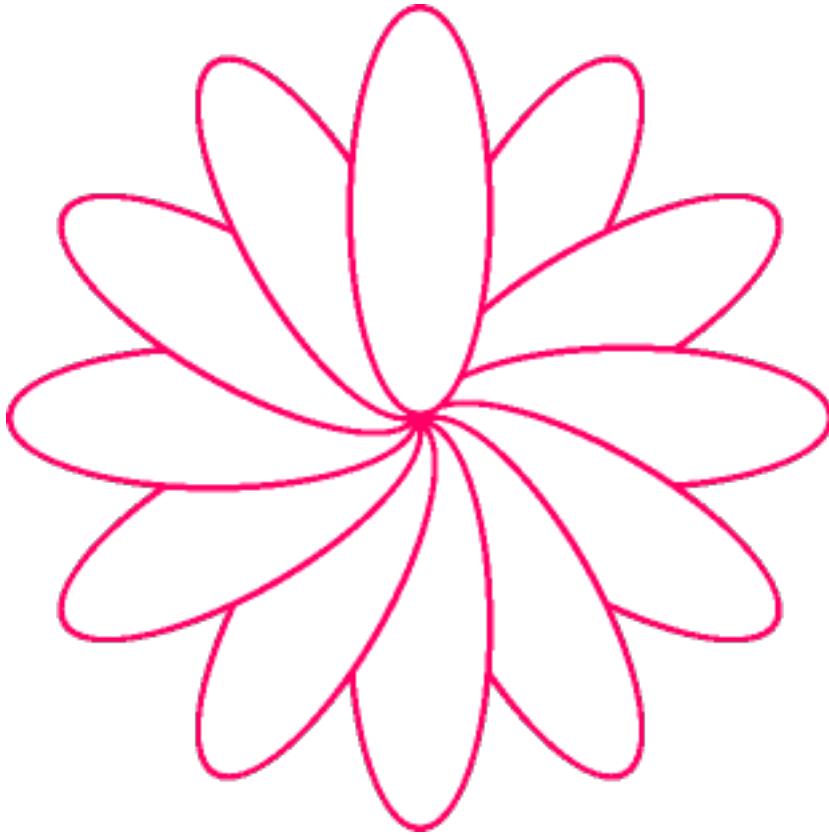
Rosa ist keine vordefinierte Farbe in PyTamaro. Sie können jedoch eine beliebige Farbe mit der Funktion `rgb_farbe(rot, gruen, blau)` erstellen. Dabei sind `rot`, `gruen` und `blau` Zahlenwerte von 0 bis 255, die den Anteil der jeweiligen Farbe an der Gesamtfarbe angeben. Damit nicht alle Kombinationen durchprobiert werden müssen, können Sie mit einem Online-Tool wie [RGB Color Picker](#) die gewünschte Farbe auswählen und die entsprechenden Werte für rot, grün und blau ablesen.

```
1 weisses_blatt = ellipse(50, 150, weiss)
2 rosa_rand = ellipse(55, 155, rgb_farbe(255, 0, 102))
3
4 bluetenblatt_mit_rand = ueberlagere(weisses_blatt, rosa_rand)
5
6 zeige_grafik(bluetenblatt_mit_rand)
```



Anschliessend können die Blütenblätter mit Rand in der vorher geschriebenen Schleife zur Gesamtgrafik zusammengefügt werden.

```
1 blutenblatt_mit_rand_fixiert = fixiere(unten_mitte, blutenblatt_mit_rand)
2
3 blume_weiss = blutenblatt_mit_rand_fixiert
4
5 for i in range(12):
6     winkel = i * 30
7     blutenblatt_gedreht = drehe(winkel, blutenblatt_mit_rand_fixiert)
8     blume_weiss = kombiniere(blume_weiss, blutenblatt_gedreht)
9
10 zeige_grafik(blume_weiss)
```



5.1.5. Schritt 5: Scheibe im Zentrum der Blume

Bei genauer Betrachtung der Blume fällt auf, dass die Scheibe in der Mitte der Blume nicht einfach einfarbig gelb ist, sondern gegen die Mitte hin einen Farbverlauf aufweist. Diesen Farbverlauf wird erzeugt, indem mehrere Kreise mit kleiner werdendem Radius und abnehmender Farbintensität übereinandergelegt werden.

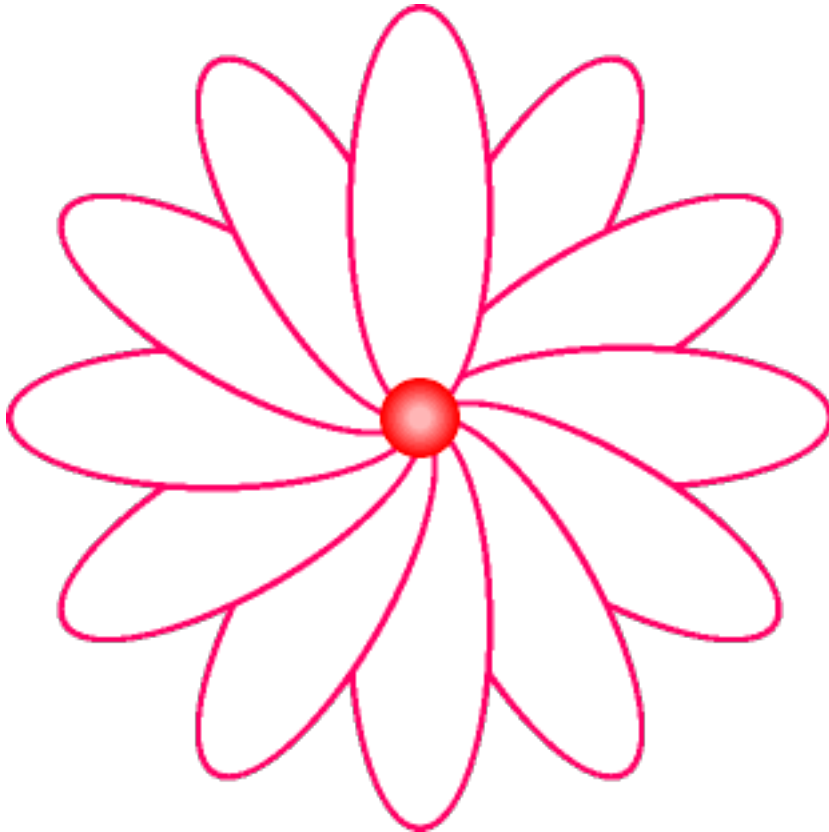
```
1  scheibe = ellipse(30, 30, rgb_farbe(255, 0, 0))
2
3  for i in range(1, 8):
4      tmp = ellipse(30 - 3*i, 30 - 3*i, rgb_farbe(255, i*26, i*26))
5      scheibe = ueberlagere(tmp, scheibe)
6
7  zeige_grafik(scheibe)
```



5.1.6. Schritt 6: Kombination der Elemente zur vollständigen Blume

Zum Schluss wird die Scheibe auf die Blütenblätter gelegt und Blume so vervollständigt.

```
1 blume_komplett = ueberlagere(scheibe, blume_weiss)
2 zeige_grafik(blume_komplett)
```



6. Programmverzweigungen (Bedingungen)

Es gibt Situationen, in denen es wünschenswert ist, dass ein Teil eines Programms nur ausgeführt wird, wenn eine bestimmte Voraussetzung erfüllt ist.

Als Beispiel dafür mag die Blume von vergangener Woche dienen. Wenn eine Blume gezeichnet werden soll, welche abwechselungsweise dunkelblaue und hellblaue Blütenblätter hat.

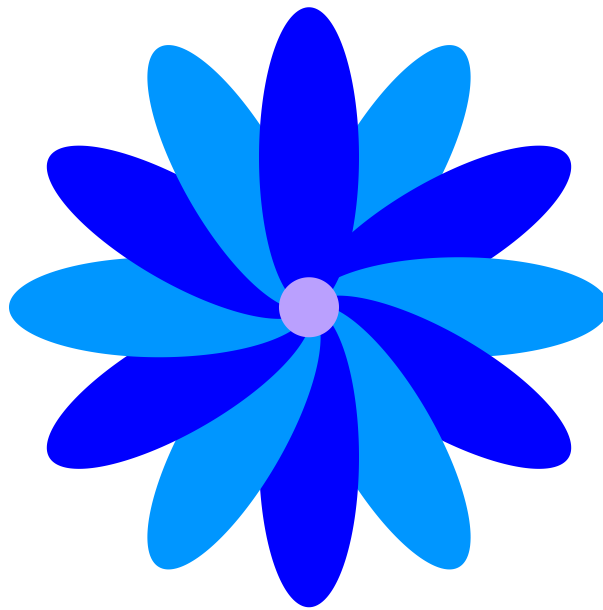


Abbildung 6.1.: Beispielblume

Diese Blume wurde ebenfalls mit einer `for`-Schleife gezeichnet. Allerdings wurde abwechselungsweise ein dunkelblaues und ein hellblaues Blütenblatt verwendet.

Abwechselungsweise kann auch als wenn eine gerade Zahl kommt bzw. wenn eine ungerade Zahl kommt verstanden werden. Kommen beim Zählen doch abwechselungsweise gerade und ungerade Zahlen. Übersetzt in die `for i in range(12)`-Schleife heisst das, immer wenn der Wert in der Variabel `i` eine gerade Zahl ist dann soll ein dunkelblaues Blütenblatt verwendet werden und immer wenn der Wert der Variabel `i` eine ungerade Zahl ist, soll ein hellblaues Blütenblatt verwendet werden.

In Python verwendet man dazu die folgende Syntax:

```

1  if BEDINGUNG:
2      ANWEISUNG(en)
3  else:
4      ANWEISUNG(en)

```

Das heisst, immer wenn die `BEDINGUNG` erfüllt ist (wenn sie `wahr`) ist, werden die eingerückten Anweisungen ausgeführt. In allen anderen Fällen werden die nach `else:` eingerückten Anweisungen ausgeführt.

Für die Zeichnung der Blume bedeutet das also, dass immer wenn `i` eine gerade Zahl ist, soll ein dunkelblaues Blütenblatt verwendet werden. Eine gerade Zahl ist dabei jede Zahl, die ohne Rest durch zwei Teilbar ist.

Ob dies der Fall ist, kann mit dem Modulo Operator (%) getestet werden (vgl. Abschnitt Vorbemerkungen: Python als Rechner.) Eine Zahl ist ohne Rest durch zwei Teilbar, wenn gilt

$$i \bmod 2 = 0$$

Übersetzt nach Python schreibt sich das als

```

1  if i % 2 == 0:

```

Für die Prüfung, ob das Resultat 0 ist, wird dabei `==` verwendet. Dies rührt daher, dass das einfache Gleichheitszeichen bereits als Operator für die Zuweisung eines Wertes zu einer Variabel besetzt war.

Der vollständige Code für die Abwechslungsweise Verwendung eines dunkelblauen bzw. eines hellblauen Blütenblattes sieht folgendermassen aus:

```

1  if i % 2 == 0:
2      blume = kombiniere(blume, bluetenblatt_dunkel)
3  else:
4      blume = kombiniere(blume, bluetenblatt_hell)

```

Falls mehr als zwei Fälle unterschieden werden müssen, lautet die Python Syntax

```

1  if BEDINGUNG:
2      ANWEISUNG(en)
3  elif BEDINGUNG:
4      ANWEISUNG(en)
5  else:
6      ANWEISUNG(en)

```

Dabei können beliebig viele `elif` Bedingungen wiederholt werden.

Als Bedingung kann nicht nur eine Gleichheit überprüft werden. Es ist auch möglich grösser `>`, grösser oder gleich `>=`, kleiner `<` und kleiner oder gleich `<=` zu prüfen.

6.1. Anwendungsübung zu Bedingungen in Python

6.1.1. Aufgabenstellung

Zeichnen Sie die folgende Blume.

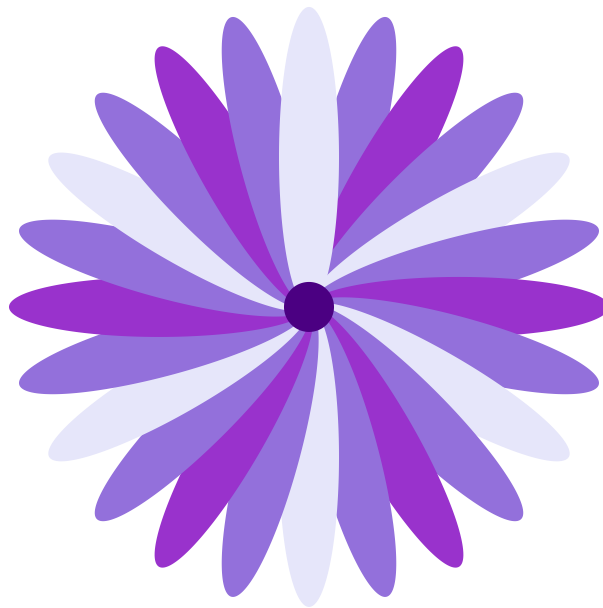


Abbildung 6.2.: Blumenvorlage

Die erforderlichen Importe und Farbdefinitionen finden Sie in der folgenden Code Zelle.

```
1 from pytamaro.de import *
2
3 lavender = rgb_farbe(230, 230, 250)
4 mediumpurple = rgb_farbe(147, 112, 219)
5 darkorchid = rgb_farbe(153, 50, 204)
6 indigo = rgb_farbe(75, 0, 130)
```



```

1 # Hier ist Platz für Ihre Blume
2 # Hinweis: die Verschiebung einer Farbe um eine Position kann erreicht
3 # werden, wenn  $i \% 4$  nicht 0 sondern eine Zahl ergibt. Bei  $\% 4$  können
4 # dies die Zahlen 1, 2 oder 3 sein.

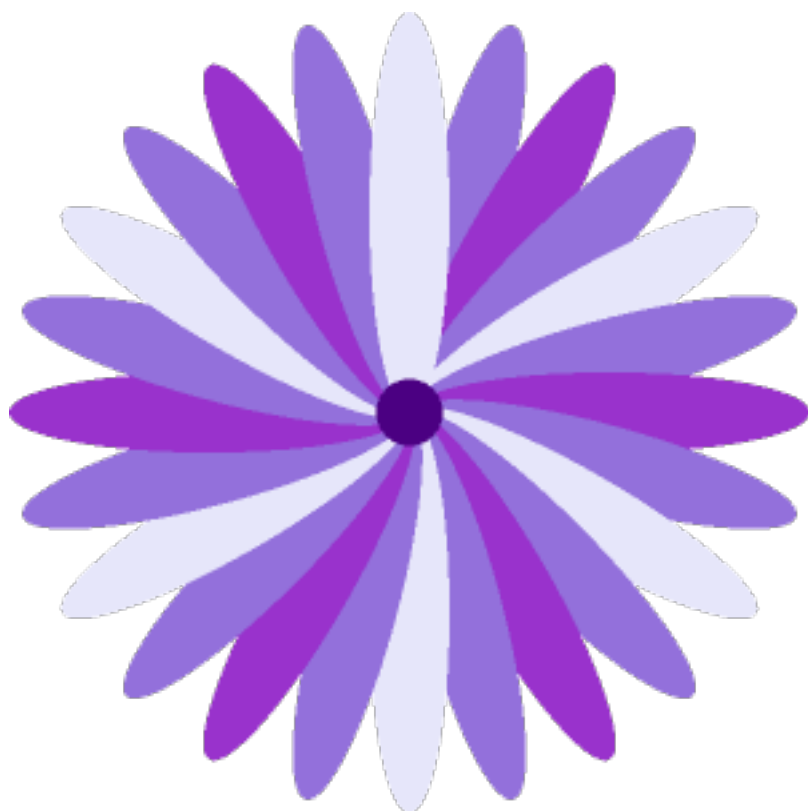
```

6.1.2. Musterlösung

```

1 blutenblatt_l = ellipse(30, 150, lavender)
2 blutenblatt_m = ellipse(30, 150, mediumpurple)
3 blutenblatt_d = ellipse(30, 150, darkorchid)
4 scheibe_i = ellipse(25, 25, indigo)
5
6 blutenblatt_m_fix = fixiere(unten_mitte, blutenblatt_m)
7 blutenblatt_l_fix = fixiere(unten_mitte, blutenblatt_l)
8 blutenblatt_d_fix = fixiere(unten_mitte, blutenblatt_d)
9
10 blume3 = scheibe_i
11 for i in range(24):
12
13     if i % 4 == 0:
14         bl_temp = drehe(i * 15, blutenblatt_l_fix)
15         blume3 = kombiniere(blume3, bl_temp)
16     elif i % 4 == 2:
17         bd_temp = drehe(i * 15, blutenblatt_d_fix)
18         blume3 = kombiniere(blume3, bd_temp)
19     else:
20         bm_temp = drehe(i * 15, blutenblatt_m_fix)
21         blume3 = kombiniere(blume3, bm_temp)
22
23 zeige_grafik(blume3)

```



7. Programmieren einfacher Funktionen

7.1. Kreisfläche

Die Fläche eines Kreises berechnet sich nach der Formel

$$A = \pi r^2$$

wobei A die Fläche π die Kreiszahl - hier mit zwei Nachkommastellen - 3.14 und r den Radius des Kreises darstellt.

Aufgabe: Implementieren Sie eine Funktion `kreisflaeche()` welche ein Argument `r` für Radius entgegennimmt und die Fläche eines Kreises zurückgibt.

```
1 def kreisflaeche():  
2     # TODO: implementieren Sie die Funktion
```

7.2. Quotient

Ein Quotient kann unterschiedlich dargestellt werden. Eine Möglichkeit ist die Darstellung als Dezimalzahl.

Aufgabe: Implementieren Sie eine Funktion `quotient()` welche zwei Dezimalzahlen als Argumente entgegennimmt und den Quotienten als Dezimalzahl zurückgibt. Sehen Sie für den Sonderfall der Division durch Null eine Lösung vor.

```
1 def quotient():  
2     # TODO: implementieren Sie die Funktion
```

7.3. Fakultät

Die Fakultät ist jene Funktion, welche das Produkt der natürlichen Zahlen bis n berechnet und wird als $n!$ geschrieben. Es gilt also

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

Wobei zu beachten ist, dass $0! = 1$ gilt.

Aufgabe: Implementieren Sie eine Funktion `fakultaet` welche als Argument $n \in \mathbb{N}$ entgegennimmt.

```
1 def fakultaet():
2     # TODO: implementieren Sie die Funktion
```

7.4. Quadratische Gleichungen (Mitternachtsformel)

Quadratische Gleichungen der Form $ax^2 + bx + c = 0$ können mit der Formel

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

gelöst werden.

Für die Implementierung ergibt sich daraus allerdings das Problem, dass die Formel zwei Resultate ergibt und eine Python Funktion nur ein Objekt zurückgeben kann.

Als Workaround können die beiden Resultate in ein Tupel gepackt werden. Ein Tupel ist eine unveränderliche Zusammenstellung mehrerer Werte. In Python werden sie in Klammern dargestellt. Die Zuweisung eines Tupels zu einer Variabel sieht dementsprechend folgendermassen aus:

```
1 t = (wert_1, wert_2, ..., wert_n)
```

Die detaillierten eigenschaften von Tupeln können hier ausser Acht gelassen werden. Es reicht x_1 und x_2 einem Tupel `resultat` zuzuweisen und `resultat` zurückzugeben.

Aufgabe: Implementieren Sie eine Funktion `mitternachtsformel`, welche die Argumente a , b und c entgegennimmt und das Tupel x_1 und x_2 zurückgibt.

```
1 def mitternachtsformel():
2     # TODO: implementieren Sie die Funktion
```

7.5. Musterlösungen

7.5.1. Kreisfläche

```
1 def kreisflaeche(r: float) -> float:
2     """
3     Berechnet die Fläche eines Kreises anhand des gegebenen Radius.
4
5     Parameter
6     -----
7     r : float
8         Der Radius des Kreises.
9
10    Rückgabewert
11    -----
12    float
13        Die Fläche des Kreises.
14    """
15    PI = 3.14 # Näherungswert für die Kreiszahl Pi
16    a = PI * r ** 2 # Formel: Fläche = Pi * r^2
17    return a
```

7.5.2. Quotient

```
1 def quotient(z: float, n: float) -> float:
2     """
3     Berechnet den Quotienten zweier Zahlen.
4
5     Gibt eine Fehlermeldung zurück, falls der Nenner null ist.
6
7     Parameter
8     -----
9     z : float
10        Der Zähler der Division.
11     n : float
12        Der Nenner der Division.
13
14     Rückgabewert
15     -----
```

```

16 float oder str
17     Der berechnete Quotient oder eine Fehlermeldung als Zeichenkette,
18     falls eine Division durch Null versucht wurde.
19     """
20
21 # Fehlermeldung für Division durch Null
22 fehlermeldung = "Eine Division durch Null ist unzulässig."
23 if n == 0:
24     return fehlermeldung # Fehlerfall: Division durch Null
25 else:
26     q = z / n # Berechnung des Quotienten
27     return q

```

7.5.3. Fakultät

```

1 def fakultaet(n: int) -> int:
2     """
3     Berechnet die Fakultät einer nicht-negativen ganzen Zahl n.
4
5     Die Fakultät n! ist das Produkt aller positiven ganzen Zahlen
6     von 1 bis n. Für n < 0 ist die Fakultät nicht definiert.
7
8     Parameter:
9         n (int): Die Zahl, deren Fakultät berechnet werden soll.
10
11     Rückgabewert:
12         int: Die Fakultät von n, oder eine Fehlermeldung als String bei n < 0.
13     """
14     if n < 0:
15         # Fakultät ist für negative Zahlen nicht definiert
16         return f'Die Fakultät für {n} ist nicht definiert.'
17     elif n == 0:
18         # Per Definition ist 0! = 1
19         return 1
20     else:
21         prod = 1 # Initialisiere das Produkt mit 1
22         for i in range(1, n + 1):
23             prod *= i # Multipliziere prod mit dem aktuellen Wert von i
24         return prod # Gib das berechnete Produkt zurück

```

7.5.4. Mitternachtsformel

```
1 def mitternachtsformel(a: float, b: float, c: float) -> tuple:
2     """
3     Löst eine quadratische Gleichung der Form  $ax^2 + bx + c = 0$ 
4     mit der Mitternachtsformel.
5
6     Parameter
7     -----
8     a : float
9         Der Koeffizient vor  $x^2$  (muss ungleich 0 sein).
10    b : float
11        Der Koeffizient vor x.
12    c : float
13        Der konstante Term.
14
15    Rückgabewert
16    -----
17    tuple oder str
18        Ein Tupel mit den beiden Lösungen (x1, x2) oder eine Fehlermeldung
19        als Zeichenkette, falls die Gleichung keine reellen Lösungen besitzt.
20    """
21    # Berechnung der Diskriminante:  $d = b^2 - 4ac$ 
22    d = b ** 2 - 4 * a * c
23    if d < 0:
24        # Keine reellen Lösungen vorhanden
25        return "Diese Gleichung hat keine Lösung."
26    else:
27        # Berechnung der beiden Lösungen mit der Mitternachtsformel:
28        #  $x_1 = \frac{-b + d^{1/2}}{2a}$ 
29        #  $x_2 = \frac{-b - d^{1/2}}{2a}$ 
30        x1 = (-b + d ** (1/2)) / (2 * a)
31        x2 = (-b - d ** (1/2)) / (2 * a)
32        return (x1, x2)
```

8. Prüfungsvorbereitung

8.1. Belegung von Variablen

Das folgende Code Snippet demonstriert die Belegung von Variablen. Insbesondere soll es auch zeigen, was geschieht, wenn die Variablen in Operationen einbezogen werden.

Überlegen Sie sich bevor Sie das Snippet ausführen zuerst Anweisung für Anweisung, welche Werte in den Variablen a , b und c nach Ausführung der jeweiligen Anweisung gespeichert sind. Notieren Sie sich die Werte auf ein Blatt Papier. Anschliessend führen Sie die Zelle zur Kontrolle Ihrer Überlegungen aus.

```
1 # Anfangswerte zuweisen
2 a = 5
3 b = 10
4 c = a
5 print(f"Nach Initialisierung:, a = {a}, b = {b}, c = {c}")
6
7 # Wert von a ändern
8 a = a + 3
9 print(f"Nach a = a + 3: a = {a}, b = {b}, c = {c}")
10
11 # Wert von b ändern
12 b = a * c
13 print(f"Nach b = a * c: a = {a}, b = {b}, c = {c}")
14
15 # Wert von c ändern
16 c = b - a
17 print(f"Nach c = b - a: a = {a}, b = {b}, c = {c}")
```

8.2. Funktionen

Die Definition einer neuen Funktion erweitert den Umfang der in Python zur Verfügung stehenden Befehle. Entsprechend ist es möglich, einmal definierte Funktionen in neuen Funktionen zu verwenden.

Unten finden Sie ein einfaches Beispiel für diese Vorgehensweise.

```
1 def erfolgsberechnung(ertrag: float, aufwand: float) -> float:
2     """Berechnet den wirtschaftlichen Erfolg als Differenz von Ertrag
3     und Aufwand.
4
5     Args:
6         ertrag: Der gesamte erzielte Ertrag.
7         aufwand: Der gesamte angefallene Aufwand.
8
9     Returns:
10        Der berechnete Erfolg (Ertrag minus Aufwand).
11    """
12    erfolg = ertrag - aufwand
13    return erfolg
14
15
16 def renditeberechnung(ertrag: float, aufwand: float,
17                       anfangskapital: float,
18                       schlusskapital: float) -> float:
19     """Berechnet die Rendite in Prozent basierend auf Erfolg
20     und durchschnittlichem Kapital.
21
22     Die Funktion nutzt die 'erfolgsberechnung', um den Erfolg
23     zu ermitteln, und setzt diesen ins Verhältnis zum durchschnittlich
24     gebundenen Kapital.
25
26     Args:
27         ertrag: Der gesamte erzielte Ertrag.
28         aufwand: Der gesamte angefallene Aufwand.
29         anfangskapital: Das Kapital zu Beginn der Periode.
30         schlusskapital: Das Kapital am Ende der Periode.
31
32     Returns:
33        Die berechnete Rendite, ausgedrückt in Prozent (%).
34    """
35    erfolg = erfolgsberechnung(ertrag, aufwand)
36    durchschnittskapital = (anfangskapital + schlusskapital) / 2
37    rendite = erfolg / durchschnittskapital * 100
38    return rendite
```

8.2.1. Berechnung des Volumens eines Zylinders

Die folgende Grafik zeigt einen Zylinder.

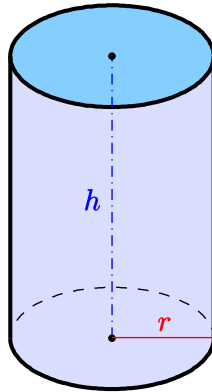


Abbildung 8.1.: Zylinder

Die Formel für die Berechnung des Volumens lautet

$$v = \pi \cdot r^2 \cdot h$$

Zerlegen Sie dazu die Berechnung in zwei Teile. Definieren Sie zuerst eine Funktion für die Berechnung der Bodenfläche und verwenden Sie diese anschliessend in einer Funktion zur Berechnung des Volumens.

Für die Kreiszahl π können Sie das Modul `math` importieren. Dies stellt Ihnen mit dem Befehl `math.pi` π in einer grossen Genauigkeit zur Verfügung.

```
1 # TODO: implementieren Sie hier Ihre Funktionen
```

8.2.2. Berechnung der kinetischen Energie eines Körpers

Die kinetische Energie eines Körpers berechnet sich nach der Formel

$$E_{kin} = \frac{1}{2} \cdot m \cdot v^2$$

wobei m für die Masse des Körpers und v für dessen Geschwindigkeit steht. Implementieren Sie eine Funktion für die Berechnung der kinetischen Energie eines Körpers. Beachten Sie dabei, dass experimentell lediglich Daten über die Masse des Körpers, die zurückgelegte Strecke des Körpers sowie die dafür erforderliche Zeit zugänglich sind.

```
1 # TODO: implementieren Sie hier Ihre Funktionen
```

8.3. Musterlösungen

8.3.1. Berechnung des Volumens eines Zylinders

```
1 import math
2
3 def kreisflaeche(r : float) -> float:
4     area = math.pi * r ** 2
5     return area
6
7 def zylindervolumen(r : float, h : float) -> float:
8     bodenflaeche = kreisflaeche(r)
9     volumen = bodenflaeche * h
10    return volumen
```

8.3.2. Berechnung der kinetischen Energie eines Körpers

```
1 def geschwindigkeit(s: float, t: float) -> float:
2     """Berechnet die Geschwindigkeit v aus zurückgelegter Strecke und
3     Zeit.
4
5     Args:
6         s: Die zurückgelegte Strecke (in Metern).
7         t: Die dafür benötigte Zeit (in Sekunden).
8
9     Returns:
10        Die berechnete Geschwindigkeit v (in m/s).
11    """
12    v = s / t
13    return v
14
15
16 def kinetische_energie(m: float, s: float, t: float) -> float:
17     """Berechnet die kinetische Energie eines Körpers basierend auf
18     experimentellen Daten.
19
```

```

20 Die Funktion berechnet zuerst die Geschwindigkeit v aus s und t
21 mithilfe der Funktion 'geschwindigkeit' und wendet dann die Formel
22  $E_{\text{kin}} = 0.5 * m * v^2$  an.
23
24 Args:
25     m: Die Masse des Körpers (in Kilogramm).
26     s: Die zurückgelegte Strecke (in Metern).
27     t: Die dafür benötigte Zeit (in Sekunden).
28
29 Returns:
30     Die berechnete kinetische Energie  $E_{\text{kin}}$  (in Joule).
31 """
32 v = geschwindigkeit(s, t)
33 e_kin = (m * v ** 2) / 2
34 return e_kin

```

9. Datenstrukturen: Listen

9.1. Daten in Listen speichern

Bisher hatten Sie Datentypen wie `int`, `float`, `str` und `bool` angeschaut. Wir möchten nun mehrere Werte in einer Liste speichern.

Ich möchte zum Beispiel alle Wochentagekürzel in einer Liste speichern. Um eine Liste im Code zu speichern, muss man die Liste mit `[` beginnen und mit `]` enden. Die Elemente werden mit einem Komma `,`

```
1 wochentage = ["Mo", "Di", "Mi", "Do", "Fr", "Sa", "So"]
2 print(wochentage)
```

9.1.1. Aufgabe: Monatsliste erstellen

Machen Sie eine Liste mit den 12 Monaten und printen Sie die Monate.

```
1 """Aufgabe: Monatsliste erstellen"""
2
3 monate = # TODO
4 print(monate)
```

9.2. Index

Die Werte werden durchnummeriert. Die dazugehörige Nummer eines Werts nennen wir den *Index*.

Das erste Element hat den Index 0!

Um ein Element aus der Liste auszugeben können wir nach dem Variablennamen die eckigen Klammern und den Index angeben.

`wochentage[0]` ist `"Mo"`. `wochentage[1]` ist `"Di"`.

9.2.1. Aufgabe: Index lesen

Lesen Sie den Codeblock und schreiben Sie hier auf, was Sie als Output erwarten, *bevor* Sie den Codeblock ausführen!

Ich erwarte, dass das Programm folgendes ausgibt:

TODO: Diese Zeile ersetzen.

```
1  liebblingstag = wochentage[3]
2  print(f"Mein Lieblingstag ist {liebblingstag}.")
```

9.3. Überschreiben von Elementen

Wir betrachten in diesem Kapitel eine Liste mit Zahlen (`int`). Wir speichern für jeden Monat die Länge des Monats. Im Falle eines Schaltjahrs möchten wir den Februar überschreiben mit 29. Studieren Sie das Beispiel:

```
1  monatslaenge = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
2  schaltjahr = True
3  if schaltjahr:
4      monatslaenge[1] = 29 # Der Februar hat den Index 2.
5  print(monatslaenge)
```

9.4. Schleifen

Listen eignen sich hervorragend um eine for-Schleife zu bauen. Betrachten Sie das Beispiel, welches den Index als `i` und den Wochentagsname als `wochentag` ausgibt. Die Liste `wochentage` möchten wir auf keinen Fall überschreiben. Also verwenden wir für die *Laufvariable* andere Variablennamen. In diesem Fall nehmen wir `i` für den Index und `wochentag` (Singular) für den Name des Tages.

```
1  for i, wochentag in enumerate(wochentage):
2      print(f"{i}: {wochentag}")
```

Falls Sie den Index nicht benötigen, können Sie die Schleife mit `for wochentag in wochentage:` machen.

9.4.1. Aufgabe: Monatsschleife komplettieren

Erstellen Sie eine Schleife, welche folgendes Ausgibt:

```
Der Januar hat 31 Tage.  
Der Februar hat 29 Tage.  
Der März hat 31 Tage.  
Der April hat 30 Tage.  
Der Mai hat 31 Tage.  
Der Juni hat 30 Tage.  
Der Juli hat 31 Tage.  
Der August hat 31 Tage.  
Der September hat 30 Tage.  
Der Oktober hat 31 Tage.  
Der November hat 30 Tage.  
Der Dezember hat 31 Tage.
```

```
1  """Aufgabe: Monatsschleife komplettieren"""  
2  
3  for ... m_name in ...:  
4      laenge = ... i ...  
5      print(f"Der {m_name} hat {laenge} Tage.")
```

9.5. Länge

Mit der Funktion `len(obj)` kann man die Länge der Liste `obj` abfragen.

```
1  """Beispiel für len(obj)"""  
2  print(f"Die Woche hat {len(wochentage)} Tage.")
```

9.6. Ergänzen und entfernen

Mit der Methode `append` kann ein Element am Ende der Liste hinzufügen und mit `pop` wieder entfernen.

```
1  tasks = ["einkaufen", "spazieren gehen", "staubsaugen"]  
2  tasks.append("essen")  
3  print(f"Neu ist auch 'essen' auf der Taskliste: {tasks}")
```

```

4
5 tasks.pop(1)
6 print(f"Welcher Task ist nun erledigt? {tasks}")
7
8 tasks.pop()
9 print(f"Wird kein Index angegeben, so wird das letzte Element in der Liste entfernt: {tasks}")

```

9.7. Aufgabe: Fibonacci

Die Fibonacci Reihe lautet 1, 1, 2, 3, 5, 8, 13, 8+13, ...

Die Summe der letzten zwei Zahlen ergeben die nächste Zahl.

Ergänzen Sie den Code, damit die Folge länger wird. Editieren Sie Stellen mit drei Punkten.

```

1 """Aufgabe: Fibonacci"""
2
3 fibonacci: list = [1, 1, 2, 3, 5, 8, 13] # Diese Zeile verändern Sie natürlich nicht.
4 for _ in range(4):
5     n = ... # Die Länge von der Folge
6     fibonacci ... (fibonacci ... + fibonacci ...)
7     print(fibonacci)

```

9.8. Aufgabe: zeichne Chomp

Wie am Anfang der Lektion mündlich besprochen kodieren wir eine Situation von [Chomp](#).



Abbildung 9.1.: Chomp_gameplay

Die dritte Situation beschreiben wir als Liste `[4, 3, 3, 3, 2]`.

Schreiben Sie eine Funktion `zeichne_chomp(chomp: list) -> None` welche die Situation zeichnet.


```

1  """Aufgabe: zeichne Chomp"""
2  from pytamaro.de import *
3
4  def zeichne_chomp(chomp: list) -> None:
5      zelle = kombiniere(
6          rechteck(68, 38, rgb_farbe(153, 103, 69)),
7          rechteck(70, 40, rgb_farbe(51, 34, 23)),
8      )
9      leere_zelle = rechteck(70, 40, weiss)
10     tafel = leere_grafik()
11     maximum = # TODO # Groesse der erste Spalte
12     for ... in chomp:
13         spalte = leere_grafik()
14         ...
15         ...
16         tafel = neben(tafel, spalte)
17     zeige_grafik(tafel)

```

```

1  # Test von der Aufgabe
2  chomp_test: list = [4, 3, 3, 3, 2]
3  zeichne_chomp(chomp_test)

```

9.9. Aufgabe: spiele Chomp

Schreiben Sie eine Funktion `chomp_spielzug(chomp: list, spielzug: list) -> list` welche aus der ersten Situation die neue Situation berechnet. Der Spielzug ist eine Liste von der Form `[x, y]`.

Test: `chomp_spielzug([4, 3, 3, 3, 2], spielzug=[3, 2])` ergibt `[4, 3, 1, 1, 1]`

Bonusidee: Überprüfen Sie, ob der Spielzug überhaupt erlaubt ist.

```

1  """Aufgabe: programmiere den Spielzug in Chomp"""
2
3  # TODO: Programmieren Sie es selber!
4
5
6  # Hier können Sie den Code testen:
7  assert chomp_spielzug([4, 3, 3, 3, 2], spielzug=[3, 2]) == [4, 3, 1, 1, 1]

```

9.10. Übersicht

Sie wissen, wie Sie

- eine Liste erstellen.
- Elemente abfragen.
- Elemente ändern.
- mit einer Schleife durch die Liste gehen.
- die Länge einer Liste bestimmen.
- Elemente hinzufügen oder entfernen.

9.11. Zusatzstoff

Weitere Infos und Features von Listen auf <https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range>

Stichwörter: Slice, Spezialfall str, insert, in, join