

PYTHON FLAG ENABLE THREE LAWS.

2. Worum geht es?

Was bedeutet es zu programmieren? **Programmieren** bedeutet, mit dem Computer zu kommunizieren. Wir teilen dem Computer mit, was er zu tun hat und zwar in einer Sprache, die er versteht. Sprachen, die der Computer versteht, nennen wir **Programmiersprachen**. Wie jede natürliche Sprache besteht auch eine Programmiersprache aus Wörtern, die eine bestimmte Bedeutung haben. Wir verwenden eine Programmiersprache, um dem Computer Anweisungen zu geben. Deshalb nennen wir die Wörter einer Programmiersprache **Computerbefehle** oder kurz **Befehle**. Manchmal wird auch das Wort **Instruktion** verwendet. Ein **Programm** besteht aus einer Reihe von Befehlen einer Programmiersprache. Das Ziel des Programmierens ist es, eine Tätigkeit zu **automatisieren**. Das bedeutet, dass wir die Ausübung einer Tätigkeit komplett dem Computer überlassen [einfach-informatik-programmieren].

Definition 2 — Imperatives Programmieren. Beim imperativen Programmieren beschreibt das Programm, **wie** man mit Computerbefehlen ein gewünschtes Ziel erreichen kann. Im Fokus steht die korrekte Reihenfolge der Befehle, um das Ziel schrittweise zu erreichen.

In diesem Skript erlernen Sie die Grundlagen der imperativen Programmierung. Wir verwenden dazu die Programmiersprache **Python** (Abbildung 2.1) in der Version 3¹.



Abbildung 2.1: Das Python-Logo.

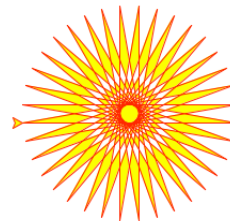


Abbildung 2.2: Beispiel für eine Turtle-Grafik.

Um die Programmiergrundlagen zu erlernen, greifen wir auf das **Turtle-Modul** von Python zurück. Mit diesem Modul bewegen wir eine Schildkröte (Turtle) mit einem „Stift“ über den Bildschirm. Wir können mit einem Python-Programm bestimmen, wie sich die Turtle bewegen soll. Abbildung 2.2 zeigt ein Beispiel für eine Grafik, welche mit der Turtle erstellt wurde.


¹Es gibt auch Python in der Version 2.

2.1 Lernziele

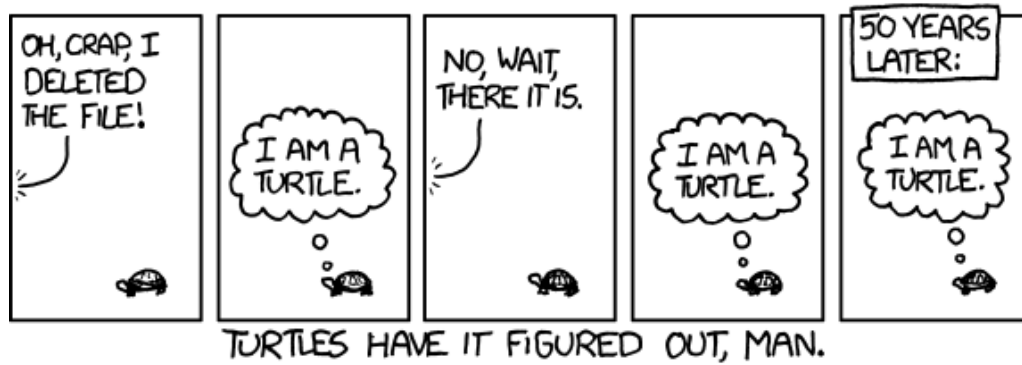
Das Skript hat folgende Ziele:

- ☐ Sie erklären, was ein Programm ist, was eine Programmiersprache ist und was es bedeutet imperativ zu programmieren.
- ☐ Sie erstellen ein Programm, welches eine gegebene geometrische Figur (z. B. ein Quadrat) zeichnet oder eine Problemstellung löst.
- ☐ Sie analysieren ein gegebenes Programm und erklären „was das Programm macht“.
- ☐ Sie erkennen Programmierfehler und korrigieren diese selbstständig.
- ☐ Sie wenden die Regeln für einen guten Programmierstil an.

Detailliertere Lernziele finden Sie in den einzelnen Kapiteln.

-  Falls es bei einem Lernziel darum geht, ein Programm zu erstellen bzw. zu analysieren, dann ist **immer** ein Programm in der Programmiersprache **Python 3** gemeint.

Alle Konzepte, die Sie mit der Turtle erlernen, können wir später auf andere Themen anwenden. Neben dem Turtle-Modul werden wir auch andere Module, welche nichts mit der Turtle „zu tun haben“, anschauen und einsetzen.



You're a turtle!

3. Mein erstes Programm

Folgende Ziele erreichen Sie nach diesem Kapitel:

- ☐ Sie erklären, was eine IDE¹ ist und warum wir eine IDE einsetzen.
- ☐ Sie erklären, was ein Ordner ist und erstellen auf Ihrem Computer einen Ordner.
- ☐ Sie erklären, was eine Datei und eine Python-Datei ist.
- ☐ Sie erstellen eine Python-Datei in einem vorgegebenen Ordner.

3.1 Quadrat zum Ersten

■ **Beispiel 3.1** Unser erstes Python-Programm wird die Turtle dazu anweisen, ein Quadrat zu zeichnen. Mit diesem Programm können Sie gleichzeitig prüfen, ob die Installation der IDE und Python geklappt hat. Erstellen Sie einen neuen Ordner mit dem Namen `01_mein_erstes_programm`. Tippen Sie dann das Listing 3.1 in eine **neue** Datei mit dem Namen `quadrat.py` ab.

```

1  import _turtle
2
3  turtle.forward(100)
4  turtle.left(90)
5  turtle.forward(100)
6  turtle.left(90)
7  turtle.forward(100)
8  turtle.left(90)
9  turtle.forward(100)
10 turtle.left(90)
11 turtle.done()
```

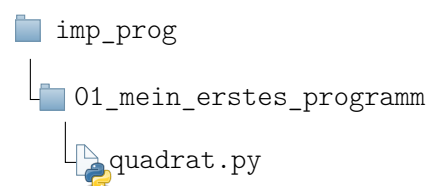


Abbildung 3.1: Datei- und Ordnerstruktur in der IDE.

Listing 3.1: Befehle für ein Quadrat (`quadrat.py`).

H Das Zeichen `_` soll verdeutlichen, dass hier ein Leerzeichen² eingegeben werden **muss**. Alle anderen Leerzeichen, welche nicht explizit abgedruckt sind, sollten aber müssen nicht notiert werden.

¹Integrated Development Environment (dt. integrierte Entwicklungsumgebung)

²Leerschlag

Die Datei- und Ordnerstruktur sollte nun wie in Abbildung 3.1 aussehen. Führen Sie abschliessend das Programm aus. Wenn Sie das Programm ausführen, dann wird das Fenster aus Abbildung 3.2 geöffnet und das Quadrat durch die „Turtle“ (das kleine Dreieck) gezeichnet.

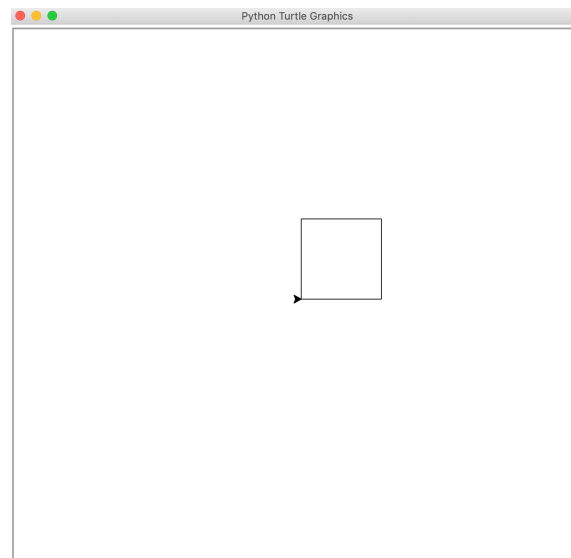


Abbildung 3.2: Resultat der Ausführung unter macOS.

■

3.2 Begriffe

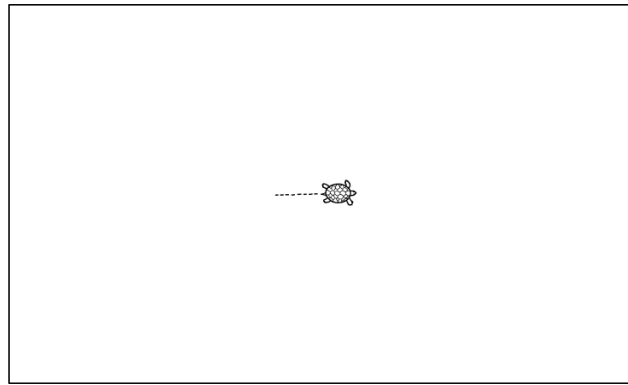
Wir klären hier zwei typische Begriffe aus dem Programmierumfeld.

Definition 3 — Listing (dt.: Programmausdruck). Bezeichnung für den vollständigen Ausdruck des Source Codes eines Programms auf Papier. Listings dienen der Dokumentation eines Programms, der Fehlersuche (grösserer Überblick über das Programm als auf dem Bildschirm oder bei der Anzeige in einem Debugger) oder zu Lehrzwecken [**def-listing**].

Definition 4 — Source Code (dt.: Quellcode). Bezeichnung für den Programmtext, der ein Programmierer bzw. eine Programmiererin eingegeben hat. Wird oft einfach mit Code abgekürzt.

3.3 Zusammenfassung

Typischerweise verwenden wir für das Erstellen eines Textes (z. B. einen Lebenslauf) ein Textverarbeitungsprogramm (z. B. Microsoft Word). Auch für das Programmieren ist es von Vorteil, wenn wir ein spezielles Programm verwenden. Solch ein Programm wird **IDE** genannt. PyCharm ist zum Beispiel eine IDE und unterstützt uns beim Programmieren mit Python. Eine Software zu erstellen, wird häufig auch als Software-Entwicklung (engl. software development) bezeichnet. In einer **Datei** (engl. file) kann ein Computeranwender Informationen langfristig auf einem Speichermedium (Festplatte, USB-Stick, Cloud etc.) speichern. Die gespeicherten Informationen können sehr vielseitig sein: Texte, Bilder, Audio, ... oder auch ein Python-Programm. Eine Datei, in der ein Python-Programm gespeichert ist, nennen wir **Python-Datei**. Eine Datei wird häufig unter Verwendung eines Punktes (.) in zwei Teile gegliedert, den eigentlichen Namen und die sogenannte **Dateinamen-Erweiterung** (engl. file extension). Für Word-Dateien gibt es zum Beispiel die Dateinamen-Erweiterung `.docx`. Python-Dateien haben die Dateinamen-Erweiterung `py`. In einem **Ordner** (engl. directory oder folder) können mehrere Dateien zusammengefasst werden. Ein Ordner kann wiederum auch einen weiteren Ordner besitzen.



Happy little turtles.

4. Turtle-Grundlagen

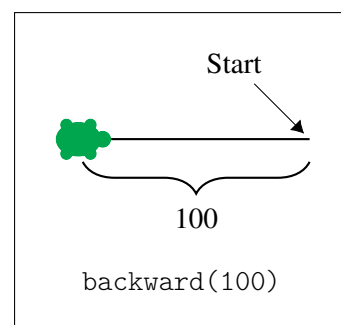
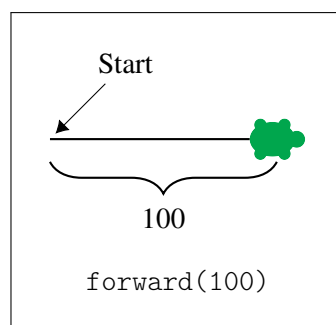
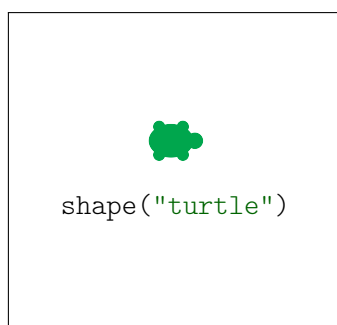
Sie kennen und verwenden nach diesem Kapitel die Python-Befehle für die folgenden Aktionen:

- ☐ Die Turtle eine Anzahl Schritte vorwärts bzw. rückwärts bewegen.
- ☐ Die Turtle um einen beliebigen Winkel (in Grad) nach rechts bzw. links drehen.
- ☐ Die Turtle in den „Wandermodus“ bzw. „Zeichenmodus“ versetzen.
- ☐ Das Icon (kleines, ausgefülltes Dreieck) anpassen.

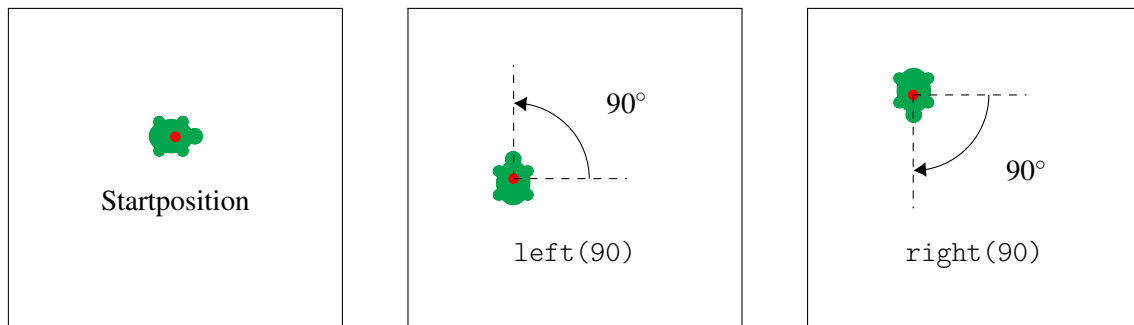
4.1 Erste Schritte mit der Turtle

Wir besprechen nun die grundlegenden Konzepte.

H Die Turtle befindet sich zu Beginn in der Mitte des Fensters und blickt nach **rechts**. Die Turtle befindet sich zu Beginn stets im **Zeichenmodus**. Ist die Turtle im Zeichenmodus, dann besitzt die Schildkröte einen Stift und zeichnet Ihren Weg auf. Dies geschieht nur, wenn die Turtle sich vorwärts oder rückwärts bewegt.



Mit `shape("turtle")` können wir das **Icon** in eine Turtle ändern. Dies hat keinen Einfluss auf die Bewegungsabläufe der Turtle. Es wird lediglich das Icon ausgetauscht. Das Icon ändert sich nur dann, wenn wir es in das Programm einbauen. Das Icon verändert sich erst nachdem die entsprechende Code-Zeile verarbeitet wurde. Mit dem Befehl `forward(distance)` bewegt sich die Turtle um `distance` Schritte vorwärts. Sie läuft dabei in **Blickrichtung**. Der Befehl `back(distance)` bewirkt das Gegenteil. Die Turtle bewegt sich um `distance` Schritte rückwärts. Sie läuft **entgegen** der **Blickrichtung**. Die Anzahl der „Schritte“ können wir frei wählen. Mit `right(angle)` können wir die Turtle um `angle` Grad nach **rechts** drehen. Den Befehl `left(angle)` können wir gleichermassen für eine Linksdrehung verwenden. Die Turtle dreht



sich dabei um `angle` Grad nach **links**. Das Drehen der Turtle bezieht sich **immer** auf die aktuelle Blickrichtung der Turtle¹. Den Winkel können wir frei wählen. Die Drehung bewegt die Turtle weder nach vorn noch zurück - es wird **nichts** gezeichnet. Prinzipiell können die Befehle zur Fortbewegung und Drehung beliebig oft und mit unterschiedlichen Zahlen in einem Programm verwendet werden.

■ **Beispiel 4.1** Das Listing 4.1 zeigt, wie wir die Turtle mit mehreren Befehlen mehrmals bewegen und drehen können. In Abbildung 4.1 wird das Resultat gezeigt.

```
1 import turtle
2
3 turtle.shape("turtle")
4 turtle.left(90)
5 turtle.forward(50)
6 turtle.right(90)
7 turtle.forward(100)
8 turtle.right(135)
9 turtle.forward(100)
10 turtle.done()
```

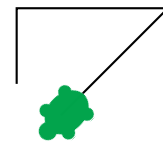


Abbildung 4.1: Resultat, wenn wir das Programm aus Listing 4.1 ausführen.

Listing 4.1: Beispielprogramm, welches die Figur aus Abbildung 4.1 zeichnet (`bsp_1.py`).

■

4.2 Programmausführung und Clean Code


Wir klären zunächst die Art und Weise, wie ein Python-Programm durch den Computer ausgeführt wird und gehen dann auf den Programmierstil ein.

Definition 5 — Programmausführung. Ein Python-Programm wird von **oben** nach **unten** ausgeführt. Der Computer liest Zeile für Zeile des Programms und führt nacheinander die entsprechenden Befehle aus. Bei einer Leerzeile passiert „nichts“. Es gibt Befehle, die Code-Zeilen überspringen oder zu einer vorherigen Code-Zeile zurückgehen.

Python führt somit ein Programm stets ab Zeile 1 aus. Damit Programme übersichtlich und leserlich bleiben, werden bei der Software-Entwicklung gewisse Regeln vereinbart. Typischerweise halten sich **alle** in einem Team an diese Regeln. Wenn wir gegenseitig Code austauschen und betrachten, findet sich somit jeder gleich zurecht. Ausserdem sollen die Regeln aufzeigen, wie wir unseren Programmierstil verbessern und dadurch verständlichere Programme erzeugen können. Wir fassen diese Regeln unter dem Begriff **Clean Code**² zusammen.

¹Manchmal hilft folgender „Trick“: Versetzen Sie sich in die Turtle und führen Sie dann die Drehung aus.

²Wikipedia fasst den Begriff „perfekt“ zusammen: https://de.wikipedia.org/wiki/Clean_Code

 **Clean Code 1 — Ein Befehl pro Zeile.** Wir notieren pro Zeile **einen** Befehl. Alle **import**-Befehle werden in den ersten Zeilen der Python-Datei notiert. Nach dem letzten **import**-Befehl fügen wir eine **Leerzeile** ein.

4.3 Wandermodus

Mit `penup()` bringen wir die Turtle in den **Wandermodus**. Im Wandermodus bewegt sich die Turtle, **ohne** mit dem Stift zu zeichnen. Der Befehl `pendown()` bringt die Turtle in den **Zeichenmodus**. Im Zeichenmodus hat die Schildkröte einen Stift und zeichnet.

■ **Beispiel 4.2** Listing 4.2 zeigt, wie wir den Wander- und Zeichenmodus verwenden.

```
1 import turtle
2
3 turtle.shape("turtle")
4 turtle.forward(50)
5 turtle.penup()
6 turtle.forward(50)
7 turtle.pendown()
8 turtle.forward(50)
9 turtle.done()
```

Listing 4.2: Beispielprogramm, welches die Figur aus Abbildung 4.2 zeichnet (`bsp_2.py`).



Abbildung 4.2: Resultat, wenn wir das Programm aus Listing 4.2 ausführen.

■

4.4 Programmierauftrag

Lesen Sie die rote Box und erfüllen Sie den Auftrag.

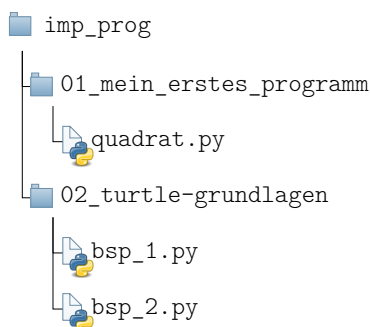
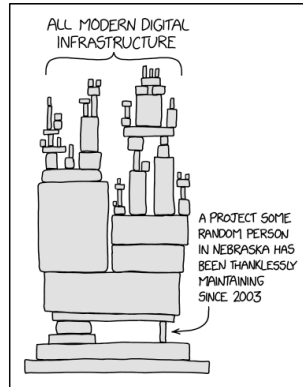


Abbildung 4.3: Ordnerstruktur in der IDE.

Wichtig! Tippen Sie den Code der beiden Listings aus diesem Kapitel ab und führen Sie das Programm aus. Erstellen Sie dazu einen neuen Ordner und zwei neue Python-Dateien. Die Datei- und Ordnerstruktur sollte nun wie in Abbildung 4.3 aussehen.

■



Someday ImageMagick will finally break for good and we'll have a long period of scrambling as we try to reassemble civilization from the rubble.

5. Import und Funktionsaufruf

Bisher haben wir beim Programmieren stets von Befehlen gesprochen. Wir haben Befehle verwendet, um die Turtle zu bewegen und zu drehen. In diesem Abschnitt geht es darum, etwas genauer hinzuschauen. Wir können Befehle in unterschiedliche Kategorie einteilen. Wir befassen uns nun mit den ersten Kategorien, analysieren die bisherigen Befehle und führen neue Fachbegriffe ein. Folgende Ziele erreichen Sie nach diesem Kapitel:

- ☐ Sie erklären die neuen Fachbegriffe und nennen dazu Beispiele.
- ☐ Sie verknüpfen die Fachbegriffe mit einem gegebenen Python-Programm.

5.1 Befehle analysieren

Wir besprechen die neuen Fachbegriffe mithilfe von Listing 5.1. Es ist das bereits bekannte Programm für ein Quadrat der Seitenlänge 100. Das Programm besteht aus zehn Befehlen, die wir zwei Kategorien zuordnen können:

- `import`-Anweisung
- Funktionsaufruf

Woher weiss ich, zu welcher Kategorie ein Befehl gehört? Dies müssen Sie in der Beschreibung zur Programmiersprache nachlesen. Wir nennen diese Beschreibung, **Dokumentation**. Die Dokumentation von Python ist unter <https://docs.python.org/> abrufbar. Wir fassen in diesen Unterlagen für ausgewählte Befehle das wichtigste aus der Dokumentation zusammen.

```
1  import turtle
2
3  turtle.forward(100)
4  turtle.left(90)
5  turtle.forward(100)
6  turtle.left(90)
7  turtle.forward(100)
8  turtle.left(90)
9  turtle.forward(100)
10 turtle.left(90)
11 turtle.done()
```

Listing 5.1: Python-Programm für ein Quadrat.

5.2 import-Anweisung

Der Befehl in der ersten Zeile aus Listing 5.1 (`import turtle`) gehört zur Kategorie „**import-Anweisung**“.

Möchten wir bestehende Programme in unserem eigenen Python-Programm verwenden, dann können wir die entsprechenden Programme hinzufügen. Dadurch erhalten wir Zugriff auf bereits existierende Befehle. Die `import`-Anweisung macht genau dies. Die bestehenden Programme werden **Module** genannt. Software-Entwickler können Module erstellen und diese zur Verfügung stellen. Jedes Modul besitzt dabei einen **Namen**. Diesen Namen geben wir bei der `import`-Anweisung an. Damit sucht Python nach dem Modul und fügt die darin enthaltenen Programmierelemente dem Programm hinzu. Die `import`-Anweisung lautet allgemein:

```
import modulname
```

Wir müssen `modulname` dann in unserem Programm durch den gewünschten Modulnamen ersetzen.

Wichtig! Achten Sie darauf, dass zwischen `import` und dem Modulnamen zwingend ein Leerzeichen eingefügt werden muss. ■

Es ist die Aufgabe des Programmierers sicherzustellen, dass der Modulname existiert.

■ **Beispiel 5.1** Unsere bisherigen Programme verwenden das Turtle-Modul. Mit `import turtle` erhalten wir die Befehle, um die Turtle zu bewegen. Es gibt noch weitere Module, die wir einsetzen werden. Wir können etwa mit `import random` Befehle verwenden, um zufällige Zahlen zu erzeugen oder mit `import math` erhalten wir Zugriff auf Befehle für mathematische Berechnungen (zum Beispiel Quadratwurzel ermitteln). ■

5.2.1 Worin liegt der Vorteil?

Häufig verwendete Programmierelemente müssen nicht selbst erstellt werden. Wir können auf bestehende Module zurückgreifen und reduzieren dadurch den Programmieraufwand.

5.3 Funktionsaufruf

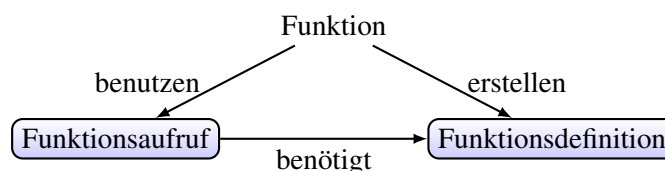
Die Befehle in den Zeilen drei bis zehn und der Befehl in Zeile zwölf aus Listing 5.1 gehören alle zur Kategorie „**Funktionsaufruf**“. Jeder dieser Befehle stellt einen Funktionsaufruf (eng. function call) dar.

Wir können eine existierende Funktion ausführen, in dem wir einen **Funktionsaufruf** verwenden. Ein Funktionsaufruf führt Code aus, der unter einem Namen zusammengefasst ist. Wir erkennen einen Funktionsaufruf wie folgt:

```
funktionsname(argumente)
```

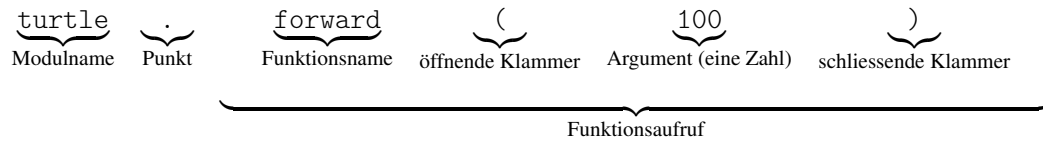
Ein Argument ist unter anderem eine Zahl oder ein Text, den wir beim Funktionsaufruf übergeben. Es gibt auch Funktionen, die kein Argument oder mehrere Argumente verlangen.

Vorerst „bedienen“ wir uns an bereits existierenden Funktionen. Wir werden später sehen, wie wir **selbst** Funktionen erstellen können (Funktionsdefinition).



Schauen wir uns nun einen bekannten Funktionsaufruf im Detail an.

■ **Beispiel 5.2** Wir rufen die Funktion `forward` mit dem Argument `100` auf. Da die Funktion aus dem Turtle-Modul stammt, müssen wir den Modulnamen vor den Funktionsaufruf notieren. Modulname und Funktionsaufruf werden durch einen **Punkt** miteinander verbunden.



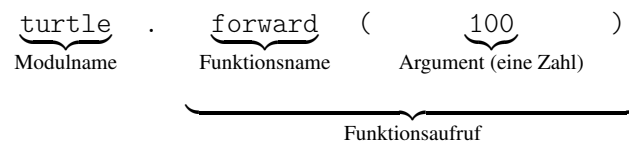
■

5.4 Aufgaben

In den folgenden Aufgaben repetieren Sie die Fachbegriffe und setzen sich mit der Python-Dokumentation auseinander.

5.4.1 Aufgabe 1

Analysieren Sie alle bisherigen Funktionsaufrufe. Beispiel:



```
turtle.backward(50)
```

```
turtle.left(90)
```

```
turtle.right(45)
```

```
turtle.pu()
```

```
turtle.pd()
```

```
turtle.done()
```

5.4.2 Aufgabe 2

1. Wer hat die Idee mit den Turtle-Grafiken erfunden und wann? Sie finden die Antwort in der Dokumentation für das Turtle-Modul. Starten Sie unter <https://docs.python.org/3/> und suchen Sie die Webseite des Turtle-Moduls.

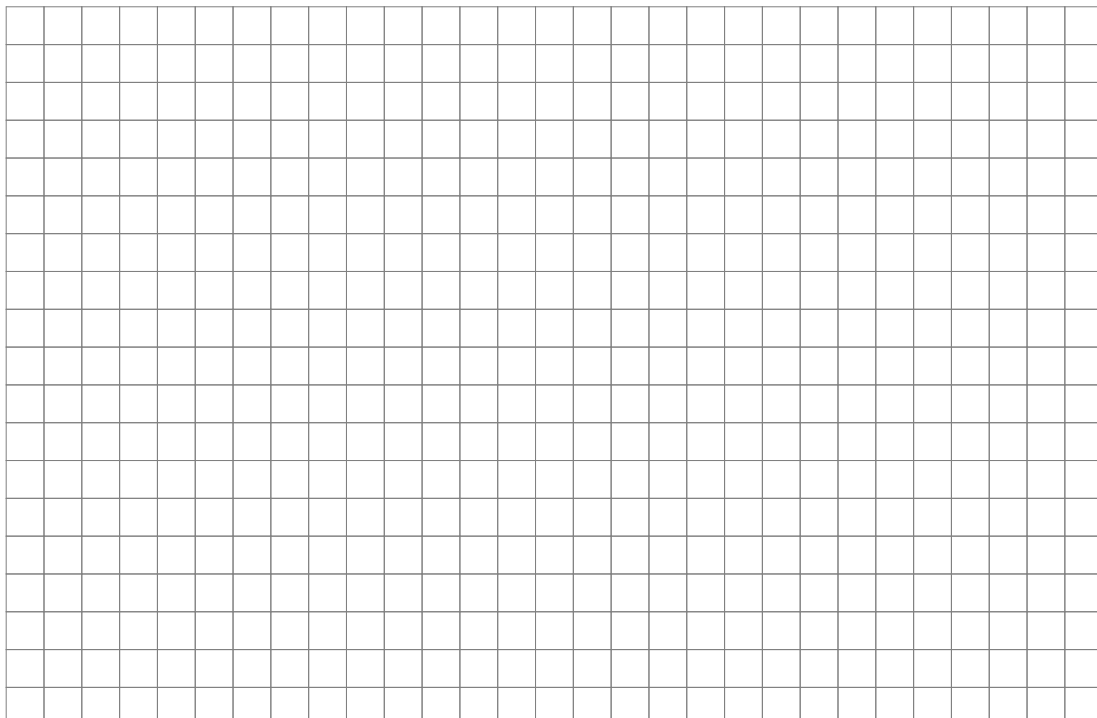
2. Welche Funktionsnamen können Sie für forward, backward, left, right, penup und pendown auch verwenden? Benutzen Sie die Dokumentation.

3. Mit welchem Funktionsaufruf können Sie die Turtle „unsichtbar“ machen? Welcher Vorteil ergibt sich dadurch? Wie können Sie die Turtle wieder anzeigen?

4. Mit welchem Funktionsaufruf können Sie die Zeichengeschwindigkeit der Turtle beeinflussen? Welche Stufen gibt es?

5.4.3 Aufgabe 3

Erstellen Sie handschriftlich ein Python-Programm, welches ein Quadrat zeichnet. Die Turtle soll nach der Hälfte des Quadrats unsichtbar werden. Stellen Sie die Zeichengeschwindigkeit der Turtle auf das Minimum (möglichst langsam zeichnen).



5.4.4 Aufgabe 4

1. Wie berechnen wir mit Python die Quadratwurzel von 42? Finden Sie die passende Funktion in der Dokumentation des Moduls `math`. Notieren Sie den Code hier.

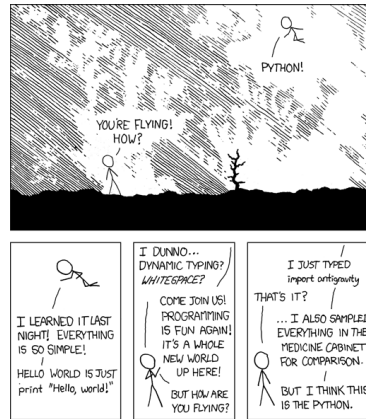
[illegible]

2. Wie erzeugen wir mit Python eine Zufallszahl zwischen 1 und 100? Finden Sie die passende Funktion in der Dokumentation des Moduls `random`. Notieren Sie den Code hier.

[illegible]

3. Wie prüfen wir mit Python, ob ein Jahr (z.B. 2017) ein Schaltjahr war? Finden Sie die passende Funktion in der Dokumentation des Moduls `calendar`. Notieren Sie den Code hier.

[illegible]



I wrote 20 short programs in Python yesterday. It was wonderful. Perl, I'm leaving you.

6. Bildschirmausgabe

Neben dem Turtle-Modul möchten wir uns Schritt-für-Schritt auch andere Bereiche anschauen, welche nichts mit der Turtle zu tun haben. Folgende Ziele erreichen Sie nach diesem Kapitel:

- ☐ Sie erstellen ein Programm, welches eine Ausgabe produziert.

6.1 Hello, World!

Wir beginnen damit, ein „Hello, World!-Programm“ zu erstellen. Dies ist fast schon eine Tradition, wenn man eine neue Programmiersprache erlernt¹. Holen wir dies also nach! Listing 6.1 zeigt dieses Programm in Python.

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

Listing 6.1: Quellcode aus `hello_world.py`.

Wenn wir dieses Programm abtippen und ausführen, dann wird der Text Hello, World! am Bildschirm angezeigt. Die Ausgabe erfolgt standardmässig im Konsolenfenster (kurz Konsole genannt).

6.2 Wie funktioniert der `print`-Befehl?

- Alles, was **zwischen** den runden Klammern steht, wird in der Konsole ausgegeben.
- Der Text `"Hello, World!"` ist das Argument für den `print`-Funktionsaufruf.
- Die geraden, doppelten Anführungszeichen werden benötigt, wenn wir in Python Text verwenden möchten. Die Anführungszeichen tauchen jedoch bei der Ausgabe **nicht** auf. Sie markieren nur den Anfang und das Ende des Textes.

Man kann den `print`-Funktionsaufruf beliebig oft benutzen. Pro `print`-Funktionsaufruf wird standardmässig eine Zeile in der Ausgabe erzeugt.

Wichtig! Der Funktionsaufruf von `print` erfolgt **ohne** ein davor notierter Modulname. Wir müssen auch kein Modul importieren, um den Funktionsaufruf einzusetzen. `print` ist eine

¹https://en.wikipedia.org/wiki/%22Hello,_World!%22_program

eingebaute Funktion (eng. built-in function). Diese Funktionsaufrufe sind **immer** und **überall** (ohne Import) durchführbar. ■

■ **Beispiel 6.1** Listing 6.2 benutzt den `print`-Funktionsaufruf zweimal.

```
1 print("Wie geht es dir?")
2 print("Mir geht es gut.")
```

Listing 6.2: Quellcode aus `print_bsp.py`.

Wie geht es dir?
Mir geht es gut.

Listing 6.3: Ausgabe in der Konsole.

H Die `print`-Funktion ermöglicht es dem **Benutzer** (\neq Programmierer) des Programms etwas mitzuteilen. Es findet also eine Kommunikation vom Computer zum Menschen statt. Der Benutzer des Programms hat typischerweise nicht Zugriff auf den Quellcode. Deshalb kann er den Text der `print`-Funktion nicht aus dem Quellcode lesen. Sie nehmen hier nun eine Doppelrolle ein. Sie sind Benutzer und Programmierer gleichzeitig!

6.3 Aufgaben

In den folgenden Aufgaben setzen Sie sich mit der `print`-Funktion auseinander.

6.3.1 Aufgabe 1

Lesen Sie den Comic oberhalb der Kapitelüberschrift. Finden Sie den „Fehler“ im Comic und korrigieren Sie den Fehler. Es ist ein „Programmierfehler“.

6.3.2 Aufgabe 2

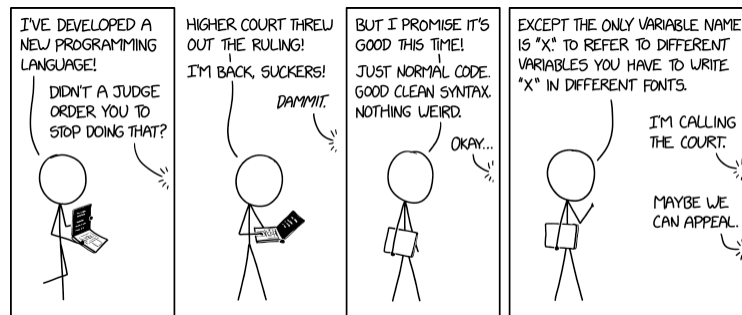
Wie muss das Python-Programm aussehen, welches folgende Ausgabe in der Konsole erzeugt?

Brauchen wir Marmelade?
Nur wenn uns die Hexe Toast serviert.

Listing 6.4: Ausgabe in der Konsole.

Notieren Sie den passenden Code hier.

[illegible]



The worst is when you run out of monospaced fonts and have to use variable-width variables.

7. Variablen

Für einfachere und vielseitigere Programme müssen wir in der Lage sein, eine Zahl oder einen Text zu speichern. In den Programmiersprachen stehen dafür **benannte Speicherplätze** zur Verfügung. Das Speichern von z.B. Zahlen oder Text dürfen wir nicht mit dem Speichern z.B. eines Textdokuments verwechseln. Ein Speicherplatz ist nur während das Programm läuft verfügbar. Danach sind alle Speicherplätze wieder gelöscht. Die Lernziele für dieses Kapitel sind:

- ☐ Sie definieren die Begriffe Variable, Wert, Literal und Zuweisung.
- ☐ Sie verwenden Variablen, um Python-Programme einfacher und vielseitiger zu gestalten.
- ☐ Sie wissen, welche Variablennamen in Python erlaubt sind.
- ☐ Sie berücksichtigen die Clean-Code-Regeln im Umgang mit Variablen.

7.1 Quadrate mit unterschiedlicher Länge

Schauen wir uns das Programm aus Listing 7.1 an, welches ein Quadrat mit der Seitenlänge 100 zeichnet. Wenn wir nun ein Quadrat mit der Seitenlänge 275 zeichnen möchten, dann müssen wir das Programm an **vier** Stellen anpassen. Listing 7.2 zeigt das angepasste Programm. Optimal wäre das Programm, wenn wir es nur an **einer** Stelle anpassen müssten. Dies können wir erreichen, in dem wir eine **Variable** verwenden.

```
1 import turtle
2
3 turtle.fd(100)
4 turtle.lt(90)
5 turtle.fd(100)
6 turtle.lt(90)
7 turtle.fd(100)
8 turtle.lt(90)
9 turtle.fd(100)
10 turtle.lt(90)
11 turtle.done()
```

Listing 7.1: Quadrat (Länge 100).

```
1 import turtle
2
3 turtle.fd(275)
4 turtle.lt(90)
5 turtle.fd(275)
6 turtle.lt(90)
7 turtle.fd(275)
8 turtle.lt(90)
9 turtle.fd(275)
10 turtle.lt(90)
11 turtle.done()
```

Listing 7.2: Quadrat (Länge 275).

Wir speichern die Zahl für die Seitenlänge des Quadrats in einer Variablen ab und verwenden die Variable anschliessend als Argument bei den vier Funktionsaufrufen. Listing 7.3 zeigt das

verbesserte Programm. Wenn das Programm ausgeführt wird, dann wird in Zeile **drei** die Zahl 100 in der Variablen mit dem Namen `a` gespeichert. In den Zeilen vier, sechs, acht und zehn wird die Variable `a` dann durch die gespeicherte Zahl ersetzt.

```
1 import turtle
2
3 a = 100
4 turtle.fd(a)
5 turtle.lt(90)
6 turtle.fd(a)
7 turtle.lt(90)
8 turtle.fd(a)
9 turtle.lt(90)
10 turtle.fd(a)
11 turtle.lt(90)
12 turtle.done()
```

Listing 7.3: Die Variable `a` speichert die Zahl 100 (`quadrat_var.py`).

```
1 import turtle
2
3 a = 275
4 turtle.fd(a)
5 turtle.lt(90)
6 turtle.fd(a)
7 turtle.lt(90)
8 turtle.fd(a)
9 turtle.lt(90)
10 turtle.fd(a)
11 turtle.lt(90)
12 turtle.done()
```

Listing 7.4: Zeile drei wurde angepasst - nun wird die Zahl 275 gespeichert.

Wir können das Programm nun bequem verändern. Wenn wir ein Quadrat mit der Seitenlänge 275 möchten, dann müssen wir nur noch Zeile drei abändern. Listing 7.4 zeigt das angepasste Programm.

7.2 Was ist eine Variable?

Eine Variable können wir uns als einen Behälter mit einer Beschriftung zur Aufbewahrung von Werten vorstellen. In Abbildung 7.1 sind die beiden Variablen `name` und `age` als Behälter dargestellt.

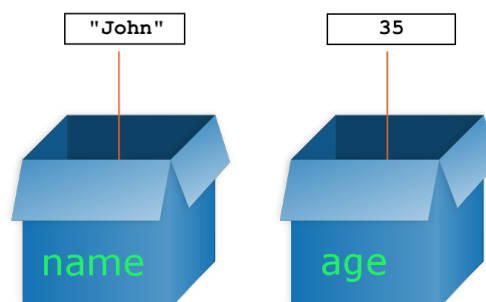


Abbildung 7.1: Zwei Variablen, dargestellt als Behälter mit Namen.¹

Jeder Behälter in Abbildung 7.1 besitzt eine Beschriftung und einen gespeicherten Wert. Der Behälter mit der Beschriftung `name` entspricht einer Variablen mit dem Namen `name`. Darin wird der Wert `"John"` (ein Text) gespeichert. Die andere Variable hat den Namen `age` und speichert den Wert 35 (eine Zahl).

Definition 6 — Variable. Eine Variable ist ein Speicherplatz mit einem Namen. In der Variablen kann ein beliebiger Wert gespeichert werden. Dies geschieht in Python mit einer **Zuweisung**. Den gespeicherten Wert können wir auch auslesen. Dazu notieren wir den Namen der Variablen. Den Namen einer Variablen können wir selbst wählen.

¹Angepasst von <http://www.pyworld.in/Python/pythonVariables.html?i=1>.

7.3 Was ist ein Wert?

Computerprogramme verarbeiten Daten. Ein **Wert** ist eines der grundlegenden Dinge, mit denen ein Programm arbeitet. Die Zahl 42 oder der Text "2001: Odyssee im Weltraum" sind zwei Beispiele für einen Wert. Werte können wir in verschiedene Kategorien aufteilen. Abbildung 7.2 zeigt eine Übersicht der bisher kennengelernten Werte.

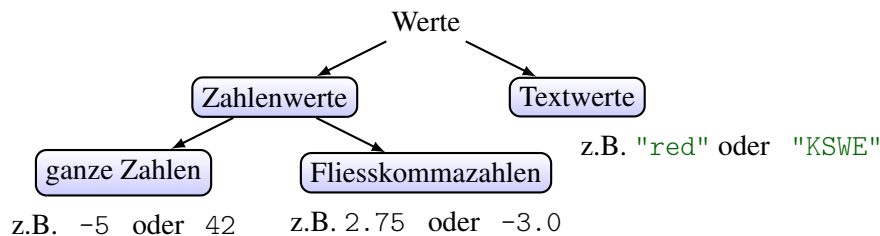


Abbildung 7.2: Diese Darstellung ist nicht abschliessend. Es gibt noch mehr Kategorien.

Einen Wert können wir **direkt darstellen**, in dem wir den Wert in das Programm eintippen.

Definition 7 — Literal. Eine Zeichenfolge, welche zur direkten Darstellung von Werten benutzt wird, nennen wir Literal.

■ **Beispiel 7.1** -5, 42, 2.75, -3.0, "red" und "KSWE" sind Beispiele für Literale. ■

Neben Literalen ist es auch möglich, durch einen **Funktionsaufruf** einen Wert zu **erzeugen**.

■ **Beispiel 7.2** Listing 7.5 und Listing 7.6 zeigen, wie wir mit einem Funktionsaufruf einen Wert erzeugen. Die Programme beinhalten neben den erzeugten Werten auch Literale (1, 101 und 42).

```

1 import random
2
3 zufallszahl = random.randrange(1, 101)
4 print(zufallszahl)

```

Listing 7.5: Erzeugt eine Zufallszahl zwischen 1 und 100 und gibt diese dann in der Konsole aus.

```

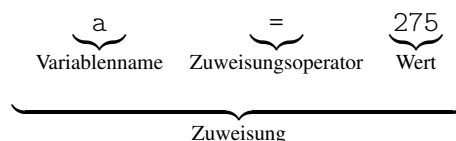
1 import math
2
3 quadratwurzel = math.sqrt(42)
4 print(quadratwurzel)

```

Listing 7.6: Berechnet die Quadratwurzel aus 42 und gibt diese dann in der Konsole aus.

7.4 Was ist eine Zuweisung?

Wir haben mit `a = 275` einen neuen Befehl kennengelernt. Der Befehl gehört zur Kategorie „Zuweisung“ (eng. assignment). `a = 275` ist somit ein Beispiel für eine **Zuweisung**.




Wir können die Zuweisung auch lesen als „a soll 275 speichern“. Eine Zuweisung erkennen wir am **Zuweisungsoperator**² (eng. assignment operator). Da das Gleichheitszeichen (=) in Python für das

²Sie kennen den Begriff bereits aus der Mathematik. Bei der Addition von zwei Zahlen, z.B. $2 + 3$, stellt das „Pluszeichen“ einen mathematischen Operator dar. Oft wird in der Mathematik auch der Begriff Rechenzeichen verwendet.

Speichern von Werten in einer Variablen benutzt wird (und nicht wie in der Mathematik im Sinne einer Gleichung), verwenden wir für das Zeichen den Fachbegriff Zuweisungsoperator.

Wichtig! Bei einer **Zuweisung** muss auf der **linken** Seite des Zuweisungsoperators eine Variable stehen. ■

 **Clean Code 2 — Leerzeichen 1.** Links und rechts eines **Zuweisungsoperators** fügen wir je ein Leerzeichen ein.

7.5 Wo wird der Inhalt einer Variablen gespeichert?

Eine Variable ist ein Speicherplatz mit einem Namen. Das Computerbauteil, in dem der Speicherplatz für eine Variable reserviert wird, heisst **Arbeitsspeicher**. Häufig wird das Bauteil auch als RAM³ bezeichnet, da für den Arbeitsspeicher im Computer fast ausschliesslich dieser Speichertyp verwendet wird. Der Arbeitsspeicher für Standardnotebooks ist typischerweise zwischen 8 GB und 32 GB gross. Nur ein **Teil** davon wird zur Speicherung von Variablen verwendet.

7.6 Welche Variablennamen sind erlaubt?

Den Namen einer Variablen können Sie fast frei wählen. Sie können Gross- und Kleinbuchstaben verwenden, Ziffern (0 – 9) und den Unterstrich⁴ (eng. underscore) (`_`). Es gibt nur sehr wenige Einschränkungen:

- Der Name darf **nicht** mit einer **Ziffer beginnen**.
- Der Name darf **nicht** einem Schlüsselwort (eng. keyword)⁵ von Python entsprechen.
- Der Name darf **keine** Leerzeichen beinhalten.

Die Namen `1farbe`, `import` und `meine Farbe` sind somit **nicht** erlaubt. `farbe1`, `warenimport` und `meineFarbe` sind hingegen erlaubt.

Wichtig! Python **unterscheidet** zwischen Gross- und Kleinbuchstaben. Wir sagen, die Programmiersprache ist **case sensitive**. ■

■ **Beispiel 7.3** Wenn Sie eine Variable `tmp`⁶ verwenden und an einer anderen Stelle `TMP` schreiben, dann sind das für Python **zwei unterschiedliche** Variablen. ■

Übrigens: Die korrekte Verwendung der Gross- und Kleinbuchstaben gilt auch für Schlüsselwörter (wie `import`).

7.7 Wie wählen wir die Variablennamen?

Eigentlich können wir recht frei einen Namen wählen, zum Beispiel `Meine_LiebliNgs_figur` oder `birne`. Jedoch **sollten** wir einen Variablennamen so wählen, dass wir durch den Namen erkennen können, was für ein Wert darin gespeichert ist. Dies entspricht der **Clean-Code-Idee**.

 **Clean Code 3 — Sinnvolle Variablennamen.** Wir wählen **Variablennamen** so, dass wir möglichst direkt verstehen, was darin gespeichert wird.

³Random Access Memory (dt. Direktzugriffsspeicher)

⁴Auch Bodenstrich oder Tiefstrich genannt.

⁵Auch reserviertes Wort genannt. Dies sind Wörter, die in Python eine bestimmte Bedeutung haben. Zum Beispiel ist `import` ein Schlüsselwort. Schlüsselwörter werden in einer IDE meist farblich hervorgehoben.

⁶Abkürzung für `temporary`.

■ **Beispiel 7.4** In Listing 7.3 haben wir den Namen `a` für die Variable gewählt. Der Name bezeichnet die Seitenlänge des Quadrats. Dies kann als ein sinnvoller Name betrachtet werden, da die Seitenlänge eines Quadrats typischerweise in der Geometrie mit a bezeichnet wird. Ausserdem wird das Programm unter dem Namen `quadrat_var.py` gespeichert. Daraus ist ersichtlich, dass es sich um die Seitenlänge handeln muss. Wir können es auch noch expliziter machen und den Namen `seitenlänge` verwenden.

Ein schlechter Name wäre `apfel`. Dieser Name sorgt für Verwirrung. Geht es hier um Obst? Was wird in einer Variablen mit dem Namen `apfel` gespeichert? Warum eine Zahl? Sollte es nicht eine Apfelsorte sein? ■

🔪 **Clean Code 4 — Snake Case.** In Python ist es üblich, für Variablenamen **nur** Kleinbuchstaben zu verwenden. Besteht der Name aus mehreren Wörtern, dann „trennen“ wir diese Wörter durch einen Unterstrich (`_`). Diese Schreibweise wird Snake Case genannt.

■ **Beispiel 7.5** Die Variablenamen `meine_farbe`, `zahl_1` oder `temperatur_in_celsius` sind in Snake Case notiert. ■

H Andere Programmiersprachen empfehlen andere Schreibweisen für Variablenamen. Abbildung 7.3 zeigt auf humorvolle Art die gebräuchlichsten Schreibweisen.

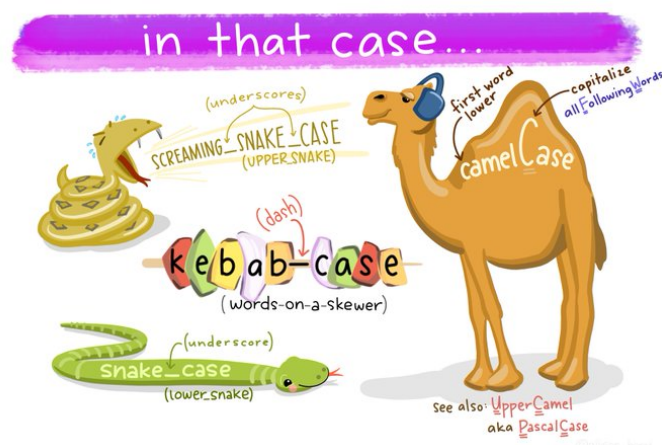


Abbildung 7.3: In Python ist „Snake Case“ üblich.

7.8 Wie benutzen wir den Inhalt einer Variablen?

Wir können jederzeit den Inhalt einer Variablen benutzen, in dem wir den entsprechenden **Variablenamen** notieren. Python ersetzt dann während der Ausführung den Variablenamen durch den gespeicherten Inhalt. Wir sagen, die Variable wird **ausgelesen**.

■ **Beispiel 7.6** Im Programm aus Listing 7.3 wird der Inhalt der Variablen `a` an vier Stellen benutzt. In Zeile vier wird für den Funktionsaufruf die Variable als Argument verwendet. Die Variable wird während der Ausführung durch die gespeicherte Zahl (100) ersetzt. Dies wiederholt sich für die Zeilen sechs, acht und zehn. ■

Wichtig! Das Auslesen einer Variable löscht den gespeicherten Inhalt der Variable **nicht**. ■

H Falls beim Auslesen die passende Variable nicht existiert (da zum Beispiel vorher kein Wert darin gespeichert wurde), dann gibt es eine **Fehlermeldung**. Typischerweise wird ein `NameError` erzeugt.

