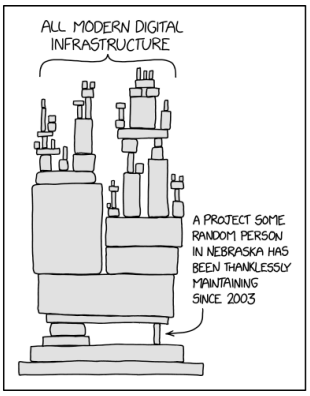
DEPENDENCY https://xkcd.com/2347/



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.

3. Analyse der Turtle-Grundlagen

Bisher haben wir beim Programmieren stets von Befehlen gesprochen. In diesem Abschnitt geht es darum, etwas genauer hinzuschauen. Wir können Befehle in verschiedene Kategorien einteilen und werden uns nun mit den ersten Kategorien beschäftigen, die bisherigen Befehle analysieren und neue Fachbegriffe einführen. Die Ziele dieses Kapitels sind:

- ☐ Sie erklären die neuen Fachbegriffe und geben dazu Beispiele.
- ☐ Sie verknüpfen die Fachbegriffe mit einem gegebenen Python-Programm.
- ☐ Sie recherchieren in der Python-Dokumentation.

3.1 Befehle analysieren

Wir besprechen die neuen Fachbegriffe anhand von Listing 4. Das Programm besteht aus zehn Befehlen, die wir in zwei Kategorien einteilen können:

- `import`-Anweisung
- Funktionsaufruf

Woher weiss ich, zu welcher Kategorie ein Befehl gehört? Diese Information müssen Sie in der Beschreibung zur Programmiersprache nachlesen. Wir bezeichnen diese Beschreibung als **Dokumentation**. Die Dokumentation für Python können wir unter `https://docs.python.org/` finden. Wir fassen in diesen Unterlagen für ausgewählte Befehle die wichtigsten Informationen 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()
12
```

Listing 4: Python-Programm für ein Quadrat.

3.2 `import`-Anweisung

Der Befehl in der ersten Zeile aus Listing 4 (`import turtle`) gehört zur Kategorie „**`import`-Anweisung**“. Dadurch erhalten wir Zugriff auf die bereits vorhandenen Befehle. Die vorhandenen Befehle werden in **Modulen** verpackt. Software-Entwickler können Module erstellen und diese zur Verfügung stellen. Jedes Modul hat einen **Namen**. Diesen Namen geben wir in der `import`-Anweisung an. Die `import`-Anweisung lautet somit im Allgemeinen:

```
import modulname
```

In unserem Programm müssen wir dann `modulname` durch den gewünschten Modulnamen ersetzen.

> **Wichtig!** Bitte beachten Sie, dass zwischen `import` und dem Modulnamen zwingend ein Leerzeichen eingefügt werden muss. ■

Es liegt in der Verantwortung des Programmierers sicherzustellen, dass der Modulname existiert.

■ **Beispiel 4** Unsere bisherigen Programme verwenden das Turtle-Modul. Mit `import turtle` erhalten wir die Befehle, um die Turtle zu bewegen. Es gibt noch andere Module, die wir verwenden 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). ■

3.2.1 Was ist der Vorteil?

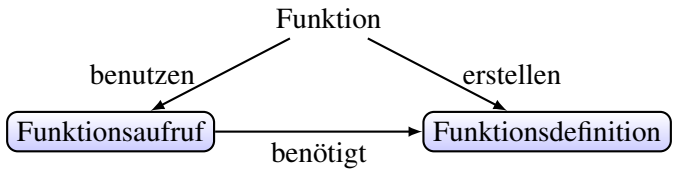
Häufig verwendete Programmierelemente müssen nicht selbst erstellt werden. Wir können auf bestehende Module zurückgreifen und reduzieren so den Programmieraufwand.

3.3 Funktionsaufruf

Die Befehle in den Zeilen drei bis zehn und der Befehl in Zeile zwölf in Listing 4 gehören alle zur Kategorie „**Funktionsaufruf**“. Jeder dieser Befehle stellt einen Funktionsaufruf (engl. function call) dar. Wir können eine vorhandene Funktion ausführen, indem wir einen **Funktionsaufruf** verwenden. Ein Funktionsaufruf führt Code aus, der unter einem Namen zusammengefasst ist. Wir erkennen einen Funktionsaufruf wie folgt:

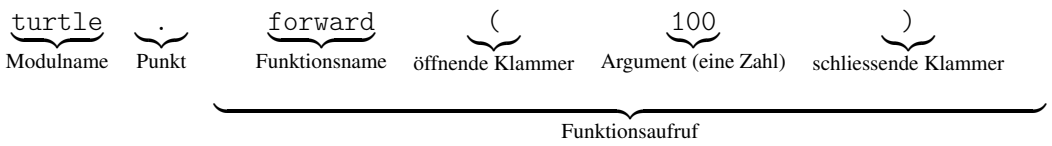
```
funktionsname(argumente)
```

Ein Argument ist z.B. eine Zahl. Manche Funktionen benötigen mehrere Argumente. Manche Funktionen brauchen gar kein Argument. Vorerst „bedienen“ wir uns an bereits existierenden Funktionen. Später werden wir sehen, wie wir **selbst** Funktionen erstellen können (Funktionsdefinition).



Betrachten wir nun einen bekannten Funktionsaufruf im Detail.

■ **Beispiel 5** Wir rufen die Funktion `forward` mit dem Argument `100` auf. Da die Funktion aus dem Turtle-Modul stammt, müssen wir den Modulnamen vor dem Funktionsaufruf notieren. Modulname und Funktionsaufruf werden durch einen **Punkt** miteinander verbunden.



■

3.3.1 Was können wir als Argument verwenden?

Wenn ein Funktionsaufruf ein Argument benötigt, dann müssen wir in der Dokumentation nachschauen, was wir als Argument verwenden können. Ein Funktionsaufruf kann nicht mit jedem Argument verwendet werden. **Argument und Funktionsaufruf müssen zusammenpassen**. Für alle Argumente haben wir bisher immer einen konkreten **Wert** verwendet: eine Zahl oder einen Text. Wir verwenden dafür die folgenden Fachbegriffe:

Definition 6 — Integer. Eine ganze Zahl, wie zum Beispiel `17`, ist ein Integer Value. Wir sagen dafür kurz: `17` ist ein **Integer**.

Definition 7 — String. Ein Text, wie zum Beispiel `"turtle"`, wird **String** genannt. In Python benötigen wir für einen String die **doppelten** Anführungszeichen.

■ **Beispiel 6** Der Funktionsaufruf `forward(100)` verwendet den Integer `100`. Beim Funktionsaufruf `shape("turtle")` wird der String `"turtle"` benutzt. ■

Clean Code — Leerzeichen 1. Mehrere **Argumente** werden durch ein **Komma** getrennt. **Nach** einem Komma notieren wir ein **Leerzeichen**.

3.4 Kommentare

Wir können einen Python-Code mit **Notizen** versehen, die für die Ausführung **keine Rolle** spielen. Diesen Vorgang nennen wir **Code kommentieren**.

Definition 8 — Python-Kommentare. Mit dem # Symbol (Hash-Symbol) startet in Python ein Kommentar. Python ignoriert bei der Ausführung alles nach dem #. Kommentare sind also für uns Menschen gedacht, die den Quellcode lesen.

Kommentare können wir direkt nach einem Befehl in derselben Zeile oder in einer separaten Zeile notieren. Wir können Kommentare auch über mehrere Zeilen verteilen. Dazu müssen wir pro Zeile ein Hash-Symbol notieren.

■ **Beispiel 7** Wir können z.B. Kommentare verwenden, um eine Code-Analyse durchzuführen.

```
1  import␣turtle
2
3  turtle.forward(100)␣␣#␣Funktionsname:␣forward,␣Argument:␣100␣(Integer)
4  turtle.left(90)
5  turtle.forward(100)
6  turtle.left(90)
7  turtle.forward(100)
8  #␣Modulname:␣turtle
9  #␣Funktionsname:␣left,␣Argument:␣90␣(Integer)
10 turtle.left(90)
11 turtle.forward(100)
12 turtle.left(90)
13 turtle.done()
14
```

Listing 5: Code-Analyse mit Kommentaren (`bsp_1.py`).

■