0 Hello world

0.0.1. Traditionell ist das erste Programm, das man in einer neuen Programmiersprache lernt, ein Programm, das «Hello world!» ausgibt.

In Python geht dies so:

```
print("Hello world!")
```

oder gleichbedeutend

```
print('Hello world!')
```

Dies illustriert auch, dass es in Python meistens egal ist, ob man einen String (= eine Zeichenkette) mit doppelten Anführungszeichen (also `"` und `"`) oder einfachen Anführungszeichen (also `'` und `'`) umschliesst.

1 Variablen, Datentypen und Datenstrukturen

1.1 Allgemeines

1.1.1. Informatik ist die Wissenschaft von der Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen. Statt von Informationen spricht man oft von Daten und meint damit vor allem die maschinenlesbare Darstellung von Informationen.

Wir erklären deswegen zunächst, wie man Daten mit Hilfe von Variablen speichert.

Definition 1.1.2 Variable (beim Programmieren)

Eine **Variable** ist ein «mit einem Namen versehener Speicherplatz für Daten».
Konkret hat eine Variable einen Namen und einen Wert von einem gewissen **Typ** (= **Datentyp**).
Der Typ gibt an, welche Art Information die Variable speichert.
Die wichtigsten Datentypen (= Mengen erlaubter Werte) sind:

- numerische Datentypen (Zahlen):
  - **Integers** = ganze Zahlen — von einer sehr kleinen, negativen bis zu einer sehr grossen, positiven ganzen Zahl
  - **Floats** = Fliesskommazahlen = Kommazahlen — bis zu einer gewissen Genauigkeit
- **Strings** = Zeichenketten
- **Booleans** = Boolescher Datentyp = Wahrheitswerte, also `False`, `True` (also wahr, falsch) (George Boole)

1.1.3. Namen von Variablen beginnen in der Regel mit einem Kleinbuchstaben, danach können weitere Kleinbuchstaben folgen, auch Grossbuchstaben, Ziffern und Unterstriche (= Bodenstriche) sind erlaubt – von der Verwendung von Umlauten und Sonderzeichen wird dringend abgeraten, Leerzeichen sind nicht erlaubt.

Variablennamen sollten kurz, aber aussagekräftig sein. Je aussagekräftiger der Variablenname ist, desto leichter tut man sich beim Programmieren und desto verständlicher sind die Programme für andere.

Anfangs ist es sinnvoll, deutsche Begriffe als Variablennamen zu verwenden: Da die Sprache Python gewisse englische Begriffe verwendet (etwa print, if, else, for, while, True, False, ...), ist dann stets klar, was man selbst definiert hat.

Beispiel 1.1.4. Wir möchten den Namen einer Person, ihre Körpergrösse, ihre letzte Mathenote und die Information, ob sie erwachsen ist, speichern.

In Python gibt es einige «built-in functions». Im obigen Programm tauchen zwei davon auf:

- `print(<Argument>)`: Gibt den Wert des Arguments im Terminal in einer Zeile aus. Es dürfen auch mehrere, durch Kommata getrennte Argumente angegeben werden; diese werden dann durch Leerzeichen getrennt ausgegeben.
- `type(<Argument>)`: Liefert den Datentyp des Arguments. Diese Funktion wird beim Programmieren sehr selten benötigt.

Merke 1.1.5 Definition von Variablen in Python, Zuweisungszeichen

Das Zeichen `=` wird in Python als **Zuweisungszeichen** verwendet.
Variablen werden in Python dadurch definiert, dass man ihren Namen links des Zuweisungszeichens aufschreibt und rechts davon den gewünschten Wert angibt.

```
<Variablenname> = <Wert>
```

Bereits definierten Variablen kann auf dieselbe Weise ein neuer Wert zugewiesen werden.

1.1.6. Variablen (in der Informatik) haben während des Programmablaufs in der Regel viele verschiedene Werte.

Dies ist ein wichtiger Unterschied zu Variablen in der Mathematik, die oft für allgemeine mathematische Objekte stehen (etwa «Sei P ein beliebiger Punkt in der Ebene.»). (Ein anderer Unterschied ist, dass Variablennamen in der Mathematik meist nur aus einem einzigen Buchstaben bestehen.)
Beispiel 1.1.7 (Forsetzung von Beispiel 1.1.4)**.** Wir nehmen an, dass unsere Person um 10 Zentimeter wächst, eine 5.5 in Mathe schreibt und erwachsen wird. Diese Datenveränderung können wir wie folgt in den Variablen speichern.

Die erste Zeile weist der bereits definierten Variablen `groesse` als neuen Wert den «alten Wert von `groesse` plus 10» zu. Beachte, dass `=` beim Programmieren ein Zuweisungszeichen ist. Die mathematische Gleichung $x = x + 10$ hingegen hat eine vollkommen andere Bedeutung (und keine Lösung).

1.2 Zahlen (integers, floats)

1.2.1 (Standardrechenzeichen)**.** Die üblichen Rechenoperationen werden in Python wie folgt geschrieben.

Operation	Symbol in Python	Beispiel in Python	mathematische Notation
Addition	`+`	`a+10`	$a + 10$
Subtraktion	`-`	`10-x`	$10 - x$
Multiplikation	`*`	`x*y`	xy oder $x \cdot y$
Division	`/`	`4/y`	$\frac{4}{y}$
Potenzieren	`**`	`a**b`	a^b

1.2.2 (Wurzeln in Python berechnen)**.** In der Mathematik lernt bzw. definiert man

$$x^{\frac{1}{2}} = \sqrt{x} \qquad x^{\frac{1}{3}} = \sqrt[3]{x} \qquad \dots \qquad x^{\frac{1}{n}} = \sqrt[n]{x}$$

Es gilt also (in einem Mischmasch aus mathematischer und Python-Notation)

$$\sqrt{x} = \text{x**0.5} = \text{x**(1/3)} \qquad \sqrt[3]{x} = \text{x**(1/2)} \qquad \sqrt[n]{x} = \text{x**(1/n)}$$

1.2.3. In Python gibt es zwei weitere Zeichen für Divisionsoperationen: Statt des oben erklärten Divisionszeichens `/` verwendet man ein «doppeltes» Divisionszeichen `//` bzw. ein Prozentzeichen = «Divisionszeichen mit zwei Kreislein» `%`.

Operation	Symbol in Python	Beispiel in Python	mathematische Notation
Rest einer Division	`%`	`63 % 15` (Ergebnis 3)	keine Standardnotation
Ganzzahlquotient	`//`	`63 // 15` (Ergebnis 4)	keine Standardnotation

Zu den Begriffen: Bei einer Division mit Rest erhält man zwei Ergebnisse: Den sogenannten **Ganzzahlquotienten** und den **Rest**. Hier ein Beispiel:

$$63 : 15 = \underbrace{4}_{\text{Ganzzahlquotient}} \text{ Rest } \underbrace{3}_{\text{Rest}} \qquad \text{gleichbedeutend} \qquad 63 = 15 \cdot 4 + 3$$

$$\text{gleichbedeutend} \qquad \frac{63}{15} = \frac{60}{15} + \frac{3}{15} = 4 + \frac{3}{15}$$

Vermeide den «beliebten Fehler»: Es gilt $\frac{63}{15} = 4.2$, aber 2 ist nicht der Rest der Division. Der Rest ist $0.2 \cdot 15 = 3$ wie soeben erklärt.

1.2.4 (Allgemeines zum Lösen von Programmieraufgaben)**.**

- Lege für jede neue Aufgabe ein neues Programm in deinem Ordner «`python`» an. Das Programm für die folgende Aufgabe A1 könntest du zum Beispiel «`rechnen-mit-variablen.py`» oder «`A1-rechnen-mit-variablen.py`» nennen.
- Das Zeichen `#` kennzeichnet in Python den Beginn eines Kommentars. Alles, was in derselben Zeile folgt, wird beim Ausführen ignoriert.

⚒ Aufgabe A1 Ergänze das folgende Programm so, dass sich der Wert der Variablen `a` so verändert, wie das in den Kommentaren in der jeweiligen Zeile angegeben ist.

Wenn du das Programm abtippst, musst du die Kommentare natürlich nicht abtippen.

```
a = 7
a = 2*a       # verdopple a
a =           # erhöhe a um 10
a =           # erhebe a in seine 3. Potenz
              # ersetze a durch seinen Ganzzahlquotient bei Division durch 7
              # erhebe a in seine 10. Potenz
              # ersetze a durch seinen Rest bei Division durch 100
              # erhöhe a um 5
              # ersetze a durch seine Quadratwurzel
print(a)
```

Zur Kontrolle deiner Lösung:
- Für `a=7` in der ersten Zeile sollte am Ende `9.0` herauskommen.
- Ändere die erste Zeile zu a=12345. Dann sollte `7.3484692283495345` ausgegeben werden.

1.2.5. Runde Klammern (also `(` und `)`) werden wie in mathematischen Termen verwendet. Auch in Python gilt die Regel «Potenzen-vor-Punkt-vor-Strich». Multiplikationspunkte (= Malpunkte) dürfen aber nicht weggelassen werden. Zum Beispiel wird $4 - 10\frac{(3^5-\sqrt{2})a}{7b}$ in Python zu `4-10*((3**5-2**(1/2))*a)/(7*b)`.

⚒ Aufgabe A2 Schreibe die folgende Formel (die von einer Variablen n abhängt) in Python-Notation in die zweite Zeile des folgenden Programms.

$$\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Bemerkung: Diese Formel ist die sogenannte Moivre-Binet-Formel zur Berechnung der n-ten Fibonacci-Zahl.

```
n = 10
f =          # hier ist die Formel in Python-Notation einzutragen
print(f)
```

Zur Kontrolle deiner Lösung:

- Für $n = 10$ liefert die obige Formel 55. Python liefert fast diese Zahl. Warum nur fast?
- Ändere die erste Zeile zu n=20. Dann ist das korrekte Ergebnis 6765.
- Die ersten n-Fibonacci-Zahlen sind $0, 1, 1, 2, 3, 5, 8, 13, 21, 35, 55, \ldots$

Aufgabe A3 In der Mathematik lernt man: Die quadratische Gleichung (in Standardform)

$$ax^2 + bx + c = 0$$

hat die beiden Lösungen (Mitternachtsformel)

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \text{und} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

(a) Du willst die Gleichung $2x^2 + 12x - 182 = 0$ mit der Mitternachtsformel in Python lösen. Tippe dafür das folgende Programm ab und korrigiere die drei falschen Zeilen.

```
a = 2
b = 12
c = -182

x1 = 1000       # zu korrigieren
x2 = 1000       # zu korrigieren

print('Die Gleichung', a, 'x^2 +', b, 'x +', c, '=0 hat die Lösungen')
print('x1=', x1)
print('x2=', x2)

print('Probe (es sollte beide Male Null herauskommen):')
# Probe für x1
print(a*x1**2+b*x1+c)
# Probe für x2
print(1000)     # zu korrigieren
```

(b) Beantworte die folgenden Fragen, indem du die ersten drei Zeilen deines Programms anpasst:

(i) Welche Lösungen hat die Gleichung $4x^2 - 28x + 49 = 0$? Was fällt dir auf?

(ii) Welche Lösungen hat die Gleichung $x^2 - 6x + 10 = 0$? Was fällt dir auf?

Aufgabe A4 Betrachte das folgende Programm

```
a = 5
b = 7
print("a =", a, ", b =", b)
#
# Hier fehlt Code
#
print("a =", a, ", b =", b)
```

Ergänze die fehlenden Code-Zeilen so, dass die Werte von `a` und `b` miteinander vertauscht werden. Die Zeilen mit den `print`-Befehlen dürfen nicht verändert werden. Das Programm soll auch dann funktionieren, wenn die Anfangswerte von `a` und `b` verändert werden.

Hinweis: Verwende eine zusätzliche Hilfsvariable `h`.

Bonusfrage: Kann man die Vertauschung auch ohne Hilfsvariable durchführen?

Aufgabe A5 Was macht der folgende Ausschnitt aus einem Programm mit den Werten der Variablen `a` und `b`? Bitte durch Nachdenken lösen, ohne Computer.

```
a = a-b
b = a+b
a = b-a
```

1.3 Strings

1.3.1. Ich erinnere daran, dass mit einem String eine Zeichenkette gemeint ist und dass Strings in Python durch Anführungszeichen (einfache oder doppelte) gekennzeichnet werden.
Beispiel 1.3.2.

```
s = 'Hello 007!'              # String
t = '12345'                   # String, der nur aus Ziffern besteht
x = 12345                     # Integer
```

1.3.3. Die Länge eines Strings kann man mit der vordefinierten Funktion `len` (für «length») wie folgt erhalten.

```
s = 'Hello!'
laenge = len(s)
print(laenge)            # Ausgabe
```

«Rechnen» mit Strings

1.3.4. In Python ist es erlaubt,

(a) zwei Strings zu «addieren»;
(b) einen Integer mit einem String zu «multiplizieren» (und auch andersherum: «String mal Integer»). ✎

```
s = 'taschen'
t = 'tuch'
print(s+t)           # Ausgabe:
print(t+s)           # Ausgabe:
print(2*s)           # Ausgabe:
print(t*3)           # Ausgabe:
print(s+t*2)         # Ausgabe:
print(len(s+t))      # Ausgabe:
print(len(s)*'-')    # Ausgabe:
```

⚒ **Aufgabe A6** Überlege dir zunächst ohne Computer, welche Ausgabe das folgende Programm liefert. Teste dann deine Vermutung mit dem Computer.

```
print("Drei" + "Sieben")
print("3 + 7")
print(3 + 7)
print(3 * "Sieben")
print(3 * 7)
print("3 * 7")
print("3" * 7)
print(3 * '7')
# und etwas schwieriger:
print(2 * "Drei-" + "Sieben!")
print(2 * ("Drei-" + "Sieben! "))
print(3 * (2 * "A" + 3 * "X" + "! "))
print("(2+" + 2 * "2*" + "2)+2")
```

1.3.5 (Zeilenumbruch)**.** Die Zeichenkombination `\n` alias «new line» in einem String sorgt bei der Ausgabe für einen Zeilenumbruch (`\n` ist ein sogenannter «escape characters»). ✎

```
print('Leitbild der Kanti:\nBegegnen\n Lernen\n  Wachsen')
# Ausgabe:
```

Aufgabe A7 Rechne mit Strings und verwende den «escape character» `\n`.

(a) Schreibe eine einzelne Code-Zeile in Python, die die folgende Ausgabe erzeugt.
Der String `"Ich liebe Informatik!"` darf darin nur einmal vorkommen.

```
Ich liebe Informatik!
Ich liebe Informatik!
Ich liebe Informatik!
```

(b) Schreibe eine weitere Code-Zeile, die die folgende Ausgabe erzeugt.
Innerhalb aller verwendeten Strings darf nur ein einziges Pluszeichen `+` vorkommen.

```
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
```

(c) Schreibe eine weitere Code-Zeile, die die folgende Ausgabe erzeugt.
Innerhalb aller verwendeten Strings dürfen nur zwei Pluszeichen `+` und zwei Minuszeichen `-` vorkommen.

```
+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+
```

Einlesen von Tastatureingaben

1.3.6. Mit der Funktion `input` können Tastatureingaben vom Benutzer entgegengenommen werden.

```
s = input('Bitte gib etwas ein! ')
print(s)
```

Der `input`-Befehl zeigt im Terminal das Argument an (hier `'Bitte gib etwas ein!'`) und wartet danach darauf, dass der Benutzer etwas eingibt und die Eingabetaste (= Return, Enter) drückt.

Die `input`-Funktion liefert als **Rückgabewert** (return value) die Eingabe des Benutzers als String. Dieser Rückgabewert wird im obigen Beispielprogramm in der Variablen `s` gespeichert und dann ausgegeben.

1.3.7. Hier ist ein fehlerhaftes Programm.

```
x = input('Gib eine ganze Zahl ein: ')
print('Das Quadrat deiner Eingabe ist')
print(x*x)                                  # Problem:
```

Lösung des Problems: Ersetze die erste Zeile des obigen Programms durch

```
x = int(input('Gib eine ganze Zahl ein: '))
```

oder durch

```
s = input('Gib eine ganze Zahl ein: ')
x = int(s)
```

1.3.8. Erklärung: Die Funktion `int` wandelt einen String in eine ganze Zahl um (sofern sinnvoll möglich).

Andere nützliche Funktionen zur Umwandlung des Datentyps sind:

- `float` von String zu Float;
- `str` von Integer oder Float zu String.

Beispiel 1.3.9.

```
s = '4.2'
print(2*s)            # Ausgabe:
s = float(s)
print(2*s)            # Ausgabe:
```

Beispiel 1.3.10.

```
x = 4.2
print(2*x)            # Ausgabe:
x = str(x)
print(2*s)            # Ausgabe:
```

Ausgabe von Variablen als Teil von Strings

1.3.11. Nicht besonders elegant:

```
x = 7
print('Die Wurzel von', x, 'ist', x**0.5, '.')
```

Variante mit Typumwandlung mit Hilfe von `str` (vermeidet z. B. das Leerzeichen vor dem Punkt am Satzende):

```
print('Die Wurzel von ' + str(x) + ' ist ' + str(x**0.5) + '.')
```

Deutlich eleganter geht es mit **f-strings** (= «formatted strings») wie folgt. Schreibe erst die gewünschte Ausgabe «naiv» wie folgt:

```
print('Die Wurzel von x ist x**0.5.')         # noch nicht korrekt
```

Dann:

(a) Schreibe den Buchstaben `f` vor das erste Anführungszeichen (f wie f-string).
(b) Klammere alle Werte, die auszugeben sind, mit geschweiften Klammern ein.

Das Ergebnis ist die folgende elegante und lesbare Variante.

```
print(f'Die Wurzel von {x} ist {x**0.5}.')
```

Beispiel 1.3.12. Hier ist ein Beispielprogramm, dass diverse oben erklärte Sachverhalte illustriert (Eingabe, Typumwandlung, Ausgabe per f-string).

Im Wesentlichen handelt es sich um eine Variante von Aufgabe A3 mit Eingabe der Parameter.

```
print("Ich kann quadratische Gleichungen der Form ax^2+bx+c=0 lösen.")
print("Gib die Werte der Parameter nacheinander ein.")

a = float(input("a="))
b = float(input("b="))
c = float(input("c="))

diskriminante = b**2-4*a*c
x1 = (-b+diskriminante**0.5)/(2*a)
x2 = (-b-diskriminante**0.5)/(2*a)

print(f'Die Gleichung {a}x^2+{b}x+{c}=0 hat die Lösungen x1={x1} und x2={x2}.')
```

(Wer sich zurecht daran stört, dass bei negativem `b` oder `c` ein Pluszeichen direkt auf ein Vorzeichen strifft, sollte `{b}` durch `({b})` ersetzen und analog für `c`. Man könnte diese Fälle auch per `if`-Bedingung abfangen.)

Dieses Programm hat noch einen Nachteil: Gibt man für den Parameter `a` Null ein, so stürzt es ab («division by zero»). Sobald wir `if`-Bedingungen kennen, ist dieser Fall einfach zu behandeln.

Aufgabe A8 Schreibe ein Programm, das vom Benutzer die Längen a und b der beiden Katheten eines rechtwinkligen Dreiecks einliest und die Länge c der Hypotenuse ausgibt. Der Dialog soll bei Eingabe von `8` und `15` wie folgt aussehen.

```
Länge der Kathete a: 8
Länge der Kathete b: 15
Die Hypotenuse hat die Länge 17.0.
```

Aufgabe A9 Vorbemerkung: Du kannst die Variable `pi` wie folgt aus der Bibliothek `math` importieren und sie dann als (Näherungswert für) π nutzen.

```
from math import pi
print(pi)              # Ausgabe: 3.141592653589793
```

Schreibe ein Programm, das vom Benutzer den Radius r eines Kreises einliest und mit Hilfe der Variablen `pi` den Umfang $U = 2\pi r$ und die Fläche $A = \pi r^2$ berechnet. Der Dialog soll bei Eingabe von `2.5` wie folgt aussehen.

```
3.141592653589793
Welchen Radius hat der Kreis? 2.5
Der Kreis mit Radius 2.5 hat den Umfang 15.707963267948966
und die Fläche 19.634954084936208.
```

Aufgabe A10 Schreibe ein Programm, das vom Benutzer einen String einliest und diesen dann in einem Rahmen aus Sternen ausgibt.

Der Dialog soll bei Eingabe von `Hello World!` wie folgt aussehen. Der Rahmen aus Sternen soll sich der Länge des eingegebenen Strings jeweils genau anpassen.

```
Gib einen String ein! Hello world!
****************
* Hello world! *
****************
```

Etwas Bonusmaterial zu f-Strings

1.3.13. Wenn man in einem f-String hinter der Variable einen Doppelpunkt schreibt, kann man danach diverse Formatierungen festlegen.Experimentiere mit dem folgenden Code (etwa Zahlen verändern) und versuche herauszufinden, welche Bedeutung die Zeichen nach dem Doppelpunkt haben. Zum Beispiel in Zeile 5: Welche Formatierung bewirkt die 20, welche die 4?

```
# Floats
x = 123**0.5
print(x)
print(f'{x:.4f}')     # das erste f steht für f-string, das zweite für Float
print(f'{x:20.4f}')

# Integers
n = 123
print(n)
print(f'{n:20d}')     # das d steht für dezimal/decimal, also Zehnersystem
print(f'{n:b}')       # das b steht für binär/binary, also Binärsystem

# Strings
s = 'Python'
print(s)
print(f'X{s:<20}X')
print(f'X{s:^20}X')
print(f'X{s:>20}X')
```