Addition

Die Aufgabe Addition war dazu da, um mit dem System und der Ein- und Ausgabe vertraut zu werden.

Nochmal als Wiederholung: Die Funktion `input()` liest eine Zeile von der Eingabe und gibt sie als String (Zeichenkette) zurück. Möchte man eine Zahl anstelle eines Strings, muss man den String mit `int(...)` umwandeln. Zusammen geschrieben wäre das `int(input())`: Hier wird zuerst eine Zeile eingelesen und dann in eine Zahl umgewandelt.

Das Resultat kann mit `print` ausgegeben werden.

```
a = int(input())
b = int(input())
print(a + b)
```

Mehrfache Addition

In Multiadder musste man die Summe von mehreren Zahlen berechnen. Wie bei Addition ging es hier darum, mit der Eingabe vertraut zu werden, diesmal mit der Eingabe von mehreren Zahlen auf einer Zeile.

Um eine Liste einzulesen kann der Trick auf dem Cheat-Sheet benutzt werden, wie in dieser Lösung auf Zeile 2:

```
1 n = int(input())
2 a = list(map(int, input().split()))
3 s = 0
4 for x in a:
5     s += x
6 print(s)
```

Mit dem Python Built-In `sum` kann man die Summe sogar direkt ausrechnen ohne Schleife:

```
1 n = int(input())
2 print(sum(map(int, input().split())))
```

FizzBuzz

Der FizzBuzz-Test ist bekannt als Frage für ein Bewerbungsgespräch um schlechte Programmierer auszusortieren.

Die Hauptschwierigkeit liegt darin, dass die Fallunterscheidungen in der falschen Reihenfolge gegeben sind. Folgender Code ist falsch, denn er gibt bei 15 nicht "`FizzBuzz`" aus, sondern nur "`Fizz`":

```
n = int(input())
for i in range(1, n+1):
    if i%3 == 0:
        print("Fizz")
    elif i%5 == 0:
        print("Buzz")
    elif i%15 == 0:
        print("FizzBuzz")
    else:
        print(i)
```

Testet man die Teilbarkeit auf 15 am Anfang, wird der Code richtig:

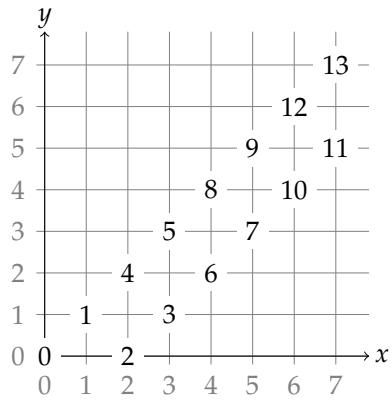
```
n = int(input())
for i in range(1, n+1):
    if i%15 == 0:
        print("FizzBuzz")
    elif i%3 == 0:
        print("Fizz")
    elif i%5 == 0:
        print("Buzz")
    else:
        print(i)
```

Die Operation % nennt man Modulo und sie gibt den Rest bei einer Division zurück. Ist der Rest 0, dann war die Zahl teilbar.

Entzückende Schrittabfolge

Sillywalk ist eher eine Mathematikaufgabe als eine Programmieraufgabe.

Es ging darum, die Regelmässigkeit in folgendem Muster zu erkennen:



Am einfachsten unterteilt man die Aufgabe in drei Fälle: Entweder befinden wir in der oberen oder in der unteren Linie, oder sind ausserhalb davon. Mit etwas ausprobieren findet man folgende Regelmässigkeit:

Obere Linie: x = y						
Schritt:	0	1	4	5	8	9
x (oder y):	0	1	2	3	4	5
$2 \cdot x$:	0	2	4	6	8	10
x%2:	0	1	0	1	0	1
$2 \cdot x - x$%2:	0	1	4	5	8	9

Untere Linie: x = y + 2						
Schritt:	2	3	6	7	10	11
$x - 1$ (oder $y + 1$):	1	2	3	4	5	6
$2 \cdot x - 2$:	2	4	6	8	10	12
x%2:	0	1	0	1	0	1
$2 \cdot x - 2 - x$%2:	2	3	6	7	10	11

Mit % wird hier die Modulo-Operation bezeichnet. Implementiert man die Fälle, kommt man auf folgenden Code:

```
x, y = map(int, input().split())
if x == y:
    print(2*x - x%2)
elif x == y+2:
    print(2*x - 2 - x%2)
else:
    print("Never")
```

Es geht aber noch ein klein wenig kürzer, mit einer Formel, die für beide Fälle funktioniert:

```
x, y = map(int, input().split())
if x == y or x == y+2:
    print(x + 2*(y//2))
else:
    print("Never")
```

Der Operator `//` steht für die Ganzzahldivision, d.h. es wird dividiert und dann abgerundet, während das normale `/` eine Kommazahl zurückgibt. Das Ergebnis von `2*(y//2)` ist deshalb nicht wieder `y`, sondern die nächstkleinere gerade Zahl; oder anders geschrieben `y-y%2`. Und das wiederum ist nichts anderes als `x-x%2` im ersten Fall, und `x-2-x%2` im zweiten Fall. Zusammen mit dem `x` kommen wir deshalb auf die gleichen Resultate wie im ersten Code.