Computerphysik Programmiertutorial 2

Prof. Dr. Matteo Rizzi und Dr. Markus Schmitt - Institut für Theoretische Physik, Universität zu Köln

ILIAS: https://www.ilias.uni-koeln.de/ilias/goto_uk_crs_3862489.html

Github: https://github.com/markusschmitt/compphys2021

Inhalt dieses Notebooks: Bemerkung zu Jupyter Notebooks, Vergleichs- und logische Operatoren, `if`-statements, `while`-Schleifen, Erster Algorithmus: Fakultät berechnen

Bemerkung zu Jupyter Notebooks

Die Zellen eines Notebooks müssen nicht unbedingt nacheinander von oben nach unten ausgeführt werden. Beliebige Ausführungsreihenfolgen sind möglich. Das bedeutet aber, dass das Ergebnis der Ausführung einer Zelle davon abhängen kann, in welcher Reihenfolge vorherige Zellen ausgeführt wurden:

In [1]:
```
x=13
```

Out[1]:
```
13
```

In [4]:
```
println("x hat den Wert $x")
```

```
x hat den Wert 27
```

In [3]:
```
x=27
```

Out[3]:
```
27
```

Vergleichs- und logische Operatoren

Wir haben bereits die Rechenoperatoren `+`, `*`, etc. kennengelernt. Eine weitere wichtige Operation ist das Vergleichen von Variablen mit

- `<` : kleiner
- `>` : größer
- `==` : gleich
- `!=` : nicht gleich
- `<=` / `>=` : kleiner oder gleich / größer oder gleich

Das Ergebnis einer Vergleichsoperation ist ein Wahrheitswert (`Bool`): `true` oder `false`.

So können wir z.B. Zahlen vergleichen:

In [5]:
```
3!=4
```

Out[5]:
```
true
```

Genauso können wir Strings vergleichen:

In [6]:
```
"petra"=="petra"
```

Out[6]:
```
true
```

Zum Programmieren ist es außerdem wichtig Wahrheitswerte logisch verknüpfen zu können, also mit Wahrheitswerten zu "rechnen".

- `&&` : logisches UND
- `||` : logisches ODER
- `!` : Negierung

In [7]:
```
true && true
```

Out[7]:
```
true
```

In [8]:
```
false || true
```

Out[8]:
```
true
```

In [9]:
```
!false
```

Out[9]:
```
true
```

In [10]:
```
true && !false
```

Out[10]:
```
true
```

In [11]:
```
(3<4) && (27>33)
```

Out[11]:
```
false
```

`if`-statements

Durch `if`-statements können wir *Verzweigungen* in unseren Algorithmus einbauen.

Syntax:

```
if <Bedingung>
    <Anweisungen>
elseif <alternative Bedingung>
    <Anweisungen>
else
    <Anweisungen>
end
```

In [16]:
```
x=11
y=11
```

Out[16]:
```
11
```

In [13]:
```
if x<y
    println("$x ist kleiner als $y")
end
```

```
10 ist kleiner als 11
```

In [17]:
```
if x<y
    println("$x ist kleiner als $y")
else
    println("$x ist *nicht* kleiner als $y")
end
```

```
11 ist *nicht* kleiner als 11
```

In [18]:
```
if x<y
    println("$x ist kleiner als $y")
elseif x==y
    println("$x ist gleich $y")
else
    println("$x ist größer als $y")
end
```

```
11 ist gleich 11
```

`while`-Schleifen

Der Computer ist nützlich, weil er uns ermöglicht ähnliche Operationen sehr schnell sehr häufig auszuführen. Dafür verwenden wir beim Programmieren **Schleifen**. Hier führen wir die `while`-Schleife ein, später werden wir auch noch die `for`-Schleife kennenlernen.

Syntax:

```
while <Bedingung>
    <Anweisungen>
end
```

In der `while`-Schleife führt der Computer die `<Anweisungen>` im Schleifenkörper (oder "body") immer wieder aus, solange die `<Bedingung>` erfüllt ist.

Achtung: Mit der `while`-Schleife kann man leicht eine `Endlosschleife` produzieren, wenn die `Bedingung` nie nicht erfüllt ist. In so einem Fall wird die Schleife endlos iteriert und die `<Anweisungen>` werden immer wieder ausgeführt.

In [19]:
```
i=0
n=10
while i<n
    i=i+1
    println("i hat jetzt den Wert $i")
end
```

```
i hat jetzt den Wert 1
i hat jetzt den Wert 2
i hat jetzt den Wert 3
i hat jetzt den Wert 4
i hat jetzt den Wert 5
i hat jetzt den Wert 6
i hat jetzt den Wert 7
i hat jetzt den Wert 8
i hat jetzt den Wert 9
i hat jetzt den Wert 10
```

Schleifen können mit dem Befehl `break` aus dem Schleifenkörper heraus abgebrochen werden:

In [20]:
```
i=0
n=10
while i<n
    i=i+1
    println("i hat jetzt den Wert $i")
    if i==7
        println("Keine Lust mehr bei i=7")
        break
    end
end
```

```
i hat jetzt den Wert 1
i hat jetzt den Wert 2
i hat jetzt den Wert 3
i hat jetzt den Wert 4
i hat jetzt den Wert 5
i hat jetzt den Wert 6
i hat jetzt den Wert 7
Keine Lust mehr bei i=7
```

Mit dem Befehl `continue` können wir zur nächsten Iteration der Schleife springen (und damit den Rest der aktuellen Iteration überspringen):

In [21]:
```
i=0
n=10
while i<n
    i=i+1
    if i==7
        println("Gib bei i=7 mal was anderes aus!")
        continue
    end
    println("i hat jetzt den Wert $i")
end
```

```
i hat jetzt den Wert 1
i hat jetzt den Wert 2
i hat jetzt den Wert 3
i hat jetzt den Wert 4
i hat jetzt den Wert 5
i hat jetzt den Wert 6
Gib bei i=7 mal was anderes aus!
i hat jetzt den Wert 8
i hat jetzt den Wert 9
i hat jetzt den Wert 10
```

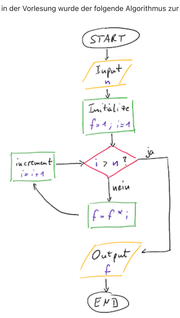
Erster Algorithmus: Fakultät berechnen

Jetzt kennen wir alle Werkzeuge um einen ersten einfachen Algorithmus zu implementieren.

Zur Erinnerung: Die Fakultät von n ist definiert als

$$n! = \prod_{j=1}^{n} j$$

und in der Vorlesung wurde der folgende Algorithmus zur Berechnung eingeführt:



In [23]:
```
n=7
```

Out[23]:
```
7
```

In [24]:
```
f=1
i=1

while !(i>n)
    f = f * i
    i = i + 1
end

println("Die Fakultät von $n ist $f")
```

```
Die Fakultät von 7 ist 5040
```

In [25]:
```
1*2*3*4*5*6*7
```

Out[25]:
```
5040
```