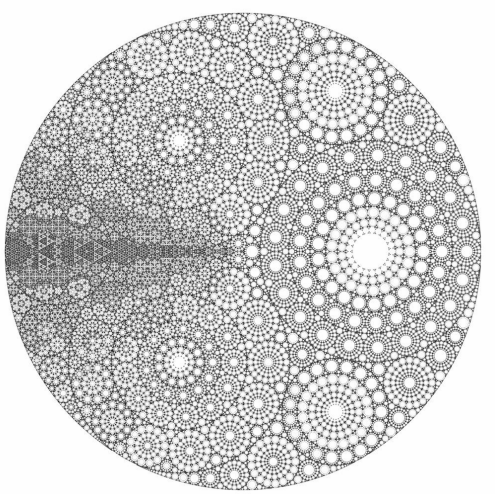
GUI + Rekursion

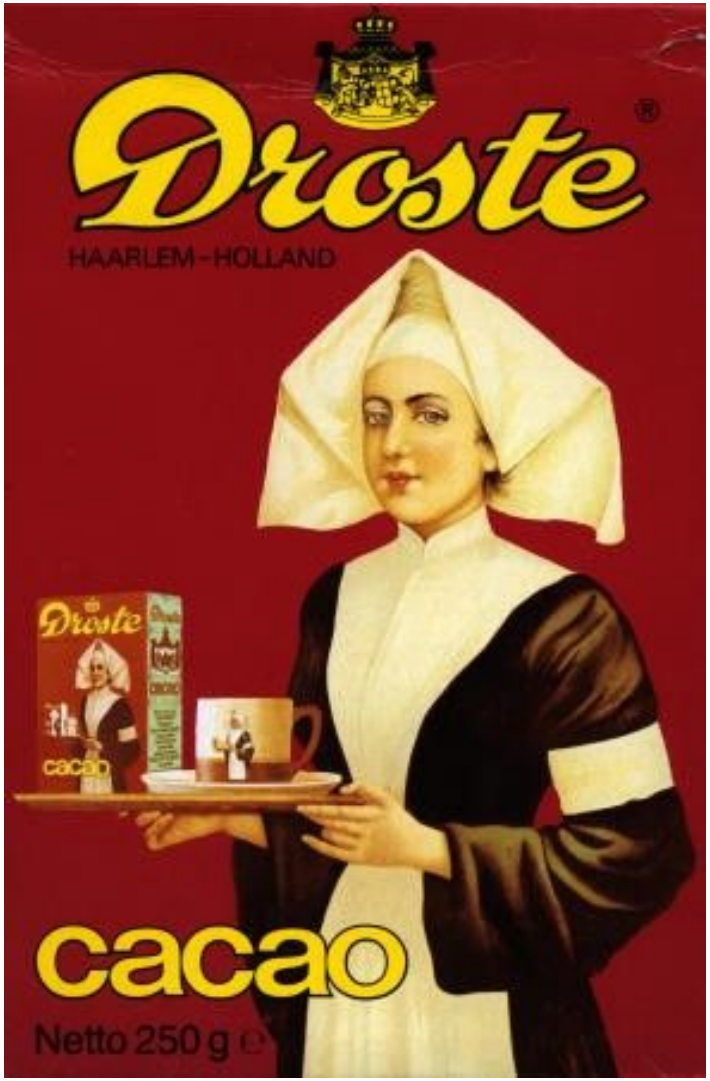


Test

- Quiz
- moodle
- https://moodle.cs.ubbcluj.ro/
- Grundlagen der Programmierung (FP IG)
- zoom

Inhalt

- pickle
- GUIs
- Rekursion
- Komplexität



Learning: Survival Guide

- ändere das Beispiel
- No Copy/Paste
  - Beispiele wiederholen
  - TIPPE DEN CODE EIN
- stelle (sinnvolle) Fragen
- schreibe aussagekräftige Suchanfragen

pickle

GUIs

- Tk toolkit
- Tkinter: die Python-Schnittstelle oder Interface zu Tk

GUIs für Python

- GUI: Graphical User Interface

- verschiedene GUIs verfügbar
  - **TKinter** - im Standard enthalten
  - PyGTk - basiert auf GTK
  - wxPython
  - PyQt - basiert auf Qt
  - PythonWin

TK

- reich an Componenten
- ausreichend für mittelgroße Anwendungen
- Widgets
  - gängiges Wort für jegliche GUI-Komponente
  - `frame` Kontainer für alles mögliche, z.B. ein Fenster
  - `canvas` Leinwand/Zeichenfläche
  - `button`
  - `checkbox`
  - `text`
  - `radiobutton`
  - `label` z.B. Text
  - `menu`
  - `scrollbar`

Hello World

```
from tkinter import *

root = Tk()
top = Toplevel()
top.mainloop()
```

Flow

```python
import tkinter as tk


class MyMainWindow:
    def __init__(self, root):
        root.title("My Window")
        root.geometry("500x500")
        self.label = tk.Label(root, text=" Hello World")
        self.label.pack()
        self.button = tk.Button(root)
        self.button.configure(text="press me")  # self.button ["text"] = "Press me"
        self.button.pack()


root = tk.Tk()
MyMainWindow(root)
root.mainloop()
```

Aktion Binding

- bestimmte Ereignisse erzeugen ein *event*
- z.B. Tastendruck, Mausklick, Mausbewegung, ...
- Ereignisse können an widgets (z.B. `Button`) gebunden werden
- Der Button wartet dann auf das jeweilige Ereignis
- tritt es auf, wird die assoziierte Funktion aufgerufen

Aktion Binding

```
import Tkinter as tk

class MyMainWindow:
    def __init__(self, root):

        root.title ("My Window")
        root.geometry("500x500")

        self.label = tk.Label (root, text = "Hello World")
        self.label.pack ()

        self.text_box = tk.Text (root, height=2, width=10)
        self.text_box.pack()

        self.add_button = tk.Button (root)
        self.add_button.configure (text = "press me")
        self.add_button.pack ()

        self.exit_button = tk.Button (root)
        self.exit_button.configure (text = "exit")
        self.exit_button.pack ()

        self.add_button.bind ("<Button -1 >", self.add_button_action)
        self.exit_button.bind ("<Button -1 >", self.exit_button_action)

    def add_button_action(self, event):
        input_text = self.text_box.get("1.0","end-1c")
        self.label.config(text=input_text)

    def exit_button_action(self, event):
        root.destroy()

root = tk.Tk ()
MyMainWindow (root)
root.mainloop ()
```

Aktion Binding

- Button reagiert nicht auf kompletten Klick
- Event `<Button-1>` ist gleich `<ButtonPress-1>`
- `1=left mouse button, 2=right, 3=middle`
- loslassen entspricht `<ButtonRelease-1>`
- mit command werden mehrere Events an ein widget gebunden

Aktion Binding

```
import Tkinter as tk

class MyMainWindow:
    def __init__(self, root):

        root.title ("My Window")
        root.geometry("500x500")

        self.label = tk.Label (root, text = "Hello World")
        self.label.pack ()

        self.text_box = tk.Text (root, height=2, width=10)
        self.text_box.pack()

        self.add_button = tk.Button (root, text = "press me",
            command = self.add_button_action)
        self.add_button.pack ()

        self.exit_button = tk.Button (root, text = "exit",
            command = self.exit_button_action)
        self.exit_button.pack ()


    def add_button_action(self):
        input_text = self.text_box.get("1.0","end-1c")
        self.label.config(text=input_text)

    def exit_button_action(self ):
        root.destroy()

root = tk.Tk ()
MyMainWindow (root)
root.mainloop ()
```

Command

- `command=self.action1`
  - `command` ist die Funktion (deren Name)

- Mit Parameter: `command=self.action1(x)`?
  - **hier wäre command der Rückgabewert der Funktion (`None`)**

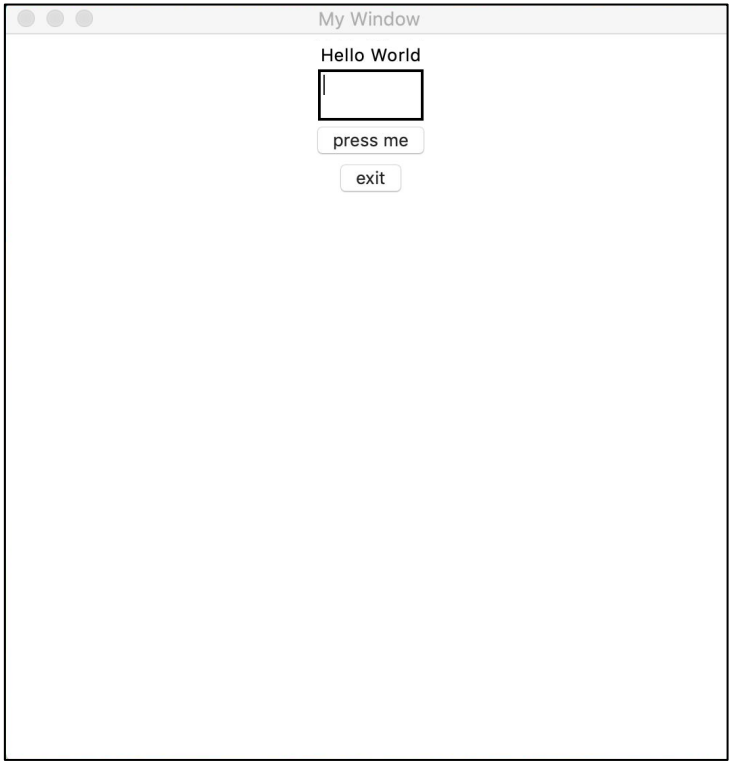
- Lösung: Funktion ohne Parameter angeben, die eine Funktion mit Parametern aufruft (`lambda-Funktion`)

```
command = lambda : self.action("do_delete"))
command = lambda : self.action1(var1 , var2 ))
command = lambda x=var1,y=var2:self.action1(x,y))
```

- Funktionsheader: `def action1(self, x, y):`

widget Hierarchie

- bisher wurden alle widgets direkt als Kinder von root erzeugt
- widgets konnen selbst Kinder haben
- beliebige Verschachtelungen möglich
- für alle widgets kann ein Padding angegeben werden
- Geometrie-Manager:
  - pack
  - grid



Geometrie-Manager

- pack
- Anordnung im wesentlichen über

```
side=... (LEFT, TOP,...)
```

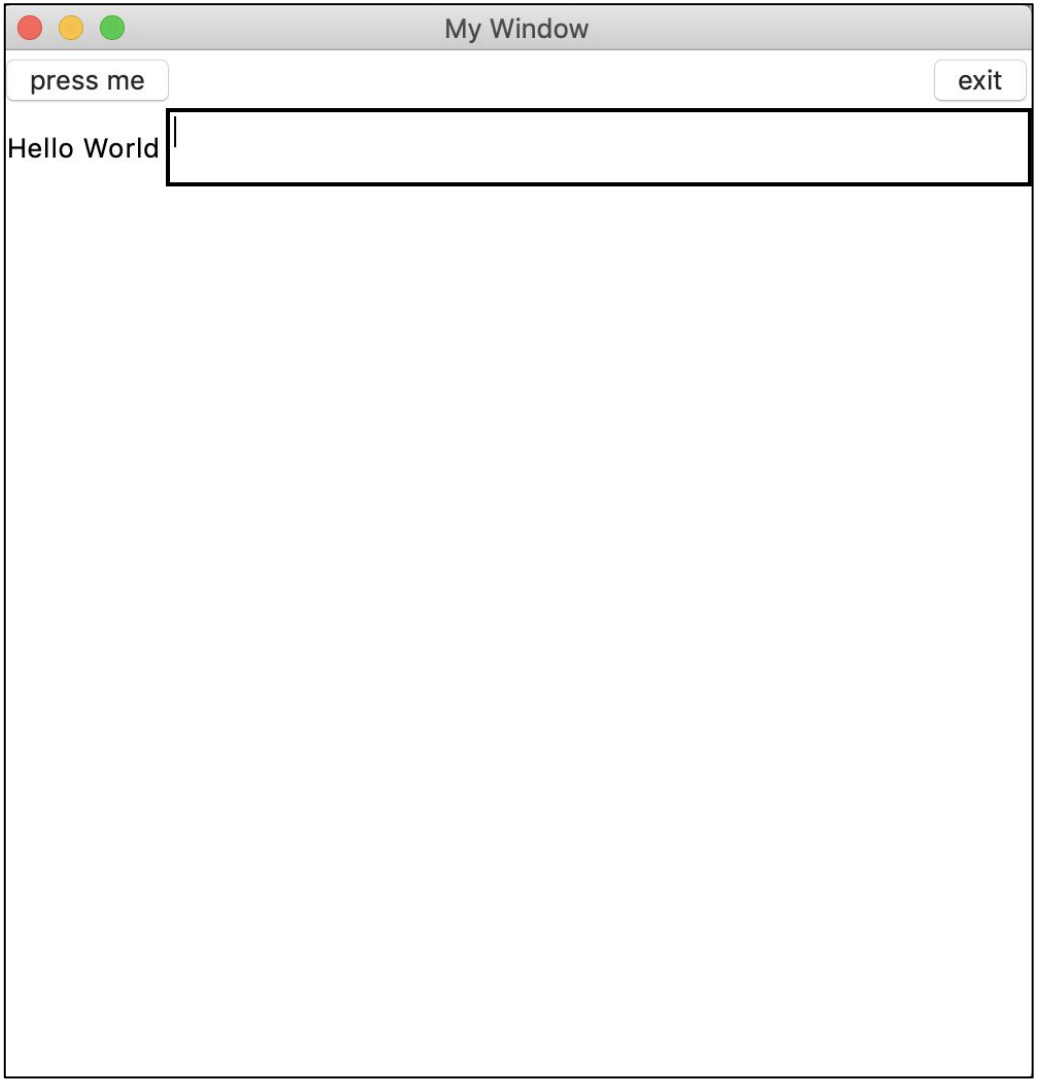
- `fill` gibt an, wie sich die Objekte ausdehnen sollen
  - `tk.NONE` gar nicht
  - `tk.X, tk.Y` nur in X/Y-Richtung
  - `tk.BOTH` in beide Richtungen
- `expand` - widget versucht, möglichst viel Fläche zu belegen
- `anchor` - damit kann das widget sich in einem bestimmten Teil der Fläche platzieren

```
tk.N, tk.NW, tk.NE, ..., tk.CENTER
```

- grid
  - Anordnung der widgets in einem Gitter
  - z.B. `widget.grid(row=2, column=7)`

Beispiel

```
import Tkinter as tk

class MyMainWindow:
    def __init__(self, root):
        root.title ("My Window")
        root.geometry("500x500")

        self.buttons_frame = tk.Frame (root)
        self.buttons_frame.pack(side = tk.TOP, fill = tk.BOTH)

        self.add_button = tk.Button (self . buttons_frame , text = "press me",
            command = self.add_button_action)
        self.add_button.pack(side = tk.LEFT)

        self.exit_button = tk.Button (self . buttons_frame , text = "exit",
            command = self.exit_button_action)
        self.exit_button.pack (side = tk.RIGHT, fill = tk.BOTH)

        self.top = tk.Frame (root)
        self.top.pack (side = tk.TOP, fill = tk.BOTH)

        self.label = tk.Label (self.top, text = "Hello World")
        self.label.pack (side = tk.LEFT)

        self.text_box = tk.Text (self.top, height=2, width=100)
        self.text_box.pack(side = tk.RIGHT)



    def add_button_action(self):
        input_text = self.text_box.get("1.0","end-1c")
        self.label.config(text=input_text)

    def exit_button_action(self ):
        root.destroy()

root = tk.Tk ()
MyMainWindow (root)
root.mainloop ()
```



Multi-Window


Wenn du einer der Non-Konformisten sein willst, dann musst du dich nur genau so anziehen wie wir und die gleiche Musik hören

Rekursion

- Neue Denkweise
- Wikipedia: “Als Rekursion bezeichnet man den Aufruf oder die Definition einer Funktion durch sich selbst.”
- Rekursion ist eine Form der Wiederholung
- Rekursion ermöglicht
  - elegante Algorithmen
  - Komplexitätsanalyse

Rekursion

In der Mathematik ist es oft einfacher, eine Funktion rekursiv zu definieren

$$\text{ggt}(a, b) = \begin{cases} a & \text{falls } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst} \end{cases}$$

- **ggt** dient zur Bestimmung des größten gemeinsamen Teilers zweier Zahlen
- Absicht deutlich aus eigener Definition

```
ggt(33, 12) = ggt(12, 33 mod 12) = ggt(12, 9)
            = ggt(9, 12 mod 9)   = ggt(9, 3)
            = ggt(3, 9 mod 3)    = ggt(3, 0)
            = 3
```

Definition

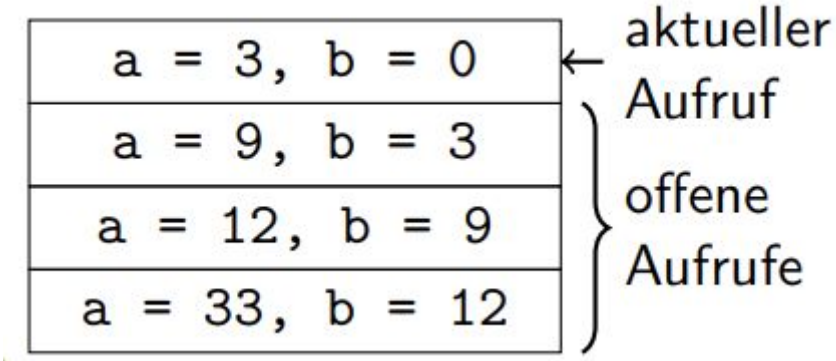
- eine Funktion nennt man **rekursiv**, wenn sie **sich selbst aufruft**
- rekursive Funktionen bestehen immer aus den folgenden
- Bestandteilen:
  - mindestens ein **Basisfall**, in dem die Rekursion abbricht und das Ergebnis entsteht
  - mindestens ein **rekursiver Fall**, in dem die Funktion sich selbst mit veränderten Argumenten aufruft

```
def ggt(a, b): //ggt = größter gemeinsamer Teiler
    if (b == 0)
        return a; //Basisfall

    return ggt(b, a % b); // rekursiver Fall
```

Rekursion und der Stack

- jeder Aufruf legt einen neuen Stack Frame mit seinen Argumenten oben auf den Stack
- die aktuellen Argumentwerte stehen im obersten Frame
- bei einem return wird der Stack Frame geschlossen

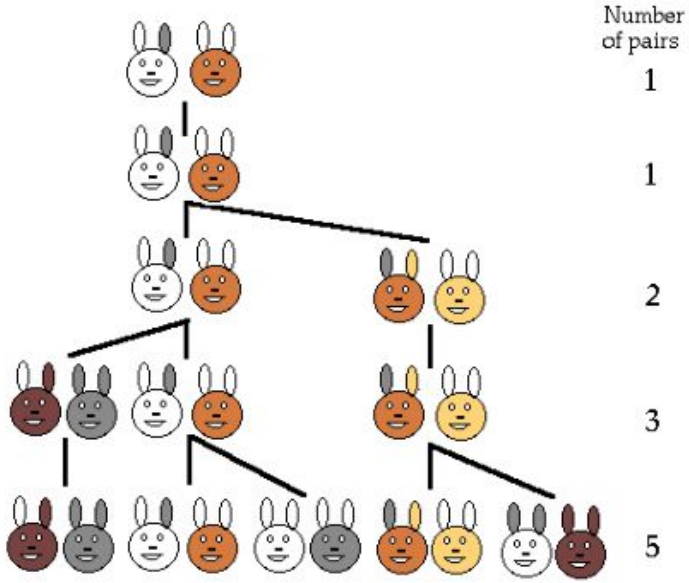
$$\begin{aligned} ggt(33, 12) &= ggt(12, 9) \\ &= ggt(9, 3) \\ &= ggt(3, 0) \\ &= 3 \end{aligned}$$



Rekursion

Beginn der Fibonacci-Folge

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...



- zu Beginn eines Jahres gibt es genau ein Paar neugeborener Kaninchen
- dieses Paar wirft nach 2 Monaten ein neues Kaninchenpaar
- und dann monatlich jeweils ein weiteres Paar
- jedes neugeborene Paar vermehrt sich auf die gleiche Weise

Fibonacci-Zahlen

- Rekursive Definition der Fibonacci-Folge:
  a. `fib(n) = 0`, falls `n = 0`
  b. `fib(n) = 1`, falls `n = 1`
  c. `fib(n) = fib(n - 1)+fib(n - 2)`, falls `n ≥ 2`
- Die Rekursion in (c) stoppt, wenn `n = 0` oder `n = 1`
- Abbruchbedingung?

```
def fibonacci(n):
    """
    compute the fibonacci number
    n - a positive integer
    return the fibonacci number for a given n
    """
    #base case
    if n==0 or n==1:
        return 1
    #inductive step
    return fibonacci(n-1)+fibonacci(n-2)
```

Fakultätsfunktion

die Fakultätsfunktion:

4! = 4 * 3 * 2 * 1

rekursive Definition der Fakultätsfunktion:

a. `faku(n) = 1`, falls `n = 0`

b. `faku(n) = n * faku(n-1)`, falls `n > 0`

```
def factorial(n):
    """
      compute the factorial
      n  is a positive integer
      return n!
    """
    if n== 0:
        return 1
    return  factorial(n-1)*n
```

Rekursion

Abbruchbedingung?

Analog zu unendlichen Schleifen mit for- and while- Schleifen

```
def gogogo(x):
    print x

    if x % 2 == 0:
        return gogogo(x / 2)
    else:
        return gogogo(3 * x + 1)
```

Rekursion vs. Iteration

- rekursive Algorithmen sind oft **natürlicher** und **einfacher** zu finden
- die **Korrektheit** rekursiver Algorithmen ist oft einfacher zu prüfen
- rekursive Lösungen sind i.d.R. statisch kürzer und auch, weil verständlicher, änderung freundlicher

Rekursion vs. Iteration

jeder rekursive Algorithmus kann in einen iterativen transformiert werden

```
TailRecursiveAlgorithm (…) {
    if condition {
        A
    } else {
        B
        TailRecursiveAlgorithm(…);
    }
}
```

```
IterativeAlgorithm (…) {
    while not condition {
        B
    }
    A
}
```

Rekursion vs. Iteration

von Iteration zu Rekursion

```
IterativeAlgorithm (…) {
    A
    while condition {
      B
    }
    C
}
```

```
Algorithm (…) {
    A
    RecursiveAlgorithm(…);
    C
}

RecursiveAlgorithm (…) {
    if condition {
      B
      RecursiveAlgorithm(…);
    }
}
```

Übung

Schreiben Sie für ein String eine Funktion, die true zurückgibt, wenn die angegebene Zeichenkette palindrom ist, andernfalls false.

Übung

Schreiben Sie ein Programm, um einen Stapel mit hilfe von Rekursion umzukehren.

Sie dürfen keine Schleifenkonstrukte wie while, for verwenden, und Sie können nur die folgenden Funktionen auf `Stack S` verwenden:

- `is_empty(S)`
- `push(S)`
- `pop(S)`

Einsatz von Rekursion

Entwurf durch Reduktion auf leichtere Probleme:

splitte ein Problem, so dass die Lösung eines bekannten einfacheren Problems auf den Rest des ursprünglichen Problems angewandt werden kann

Divide&Conquer Strategie:

teile das Problem in zwei Teilen auf und behandle jede Teile mit dem gleichen Verfahren, bis die Teile klein genug sind, um eine direkte Lösung zu erlauben

Ersetze Rekursion durch Schleifen,

wann immer dies einfach durchzuführen ist

Komplexität

Beurteilung von Algorithmen

- viele Algorithmen, um dieselbe Funktion zu realisieren
- Welche Algorithmen sind die besseren?
- nicht-funktionaler Eigenschaften:
  - Zeiteffizienz: Wie lange dauert die Ausführung?
  - Speichereffizienz: Wie viel Speicher wird zur Ausführung benötigt?
  - Benötigte Netzwerkbandbreite
  - Einfachheit des Algorithmus
  - Aufwand für die Programmierung

Ressourcenbedarf

- Prozesse verbrauchen:
  - Rechenzeit
  - Speicherplatz

- Die Ausführungszeit hängt ab von:
  - der konkreten Programmierung
  - Prozessorgeschwindigkeit
  - Programmiersprache
  - Qualität des Compilers

Beispiel

```
def fibonacci(n):
    """
     compute the fibonacci number
     n - a positive integer
     return the fibonacci number for a given n
    """
    #base case
    if n==0 or n==1:
        return 1
    #inductive step
    return fibonacci(n-1)+fibonacci(n-2)
```

```
def fibonacci2(n):
    """
     compute the fibonacci number
     n - a positive integer
     return the fibonacci number for a given n
    """
    sum1 = 1
    sum2 = 1
    rez = 0
    for i in range(2, n+1):
        rez = sum1+sum2
        sum1 = sum2
        sum2 = rez
    return rez
```

```
def measureFibo(nr):
    sw = StopWatch()
    print "fibonacci2(", nr, ") =", fibonacci2(nr)
    print "fibonacci2 take " +str(sw.stop())+" seconds"

    sw = StopWatch()
    print "fibonacci(", nr, ") =", fibonacci(nr)
    print "fibonacci take " +str(sw.stop())+" seconds"


measureFibo(32)
```

```
fibonacci2( 32 ) = 3524578
fibonacci2 take 0.0 seconds
fibonacci( 32 ) = 3524578
fibonacci take 1.7610001564 seconds
```

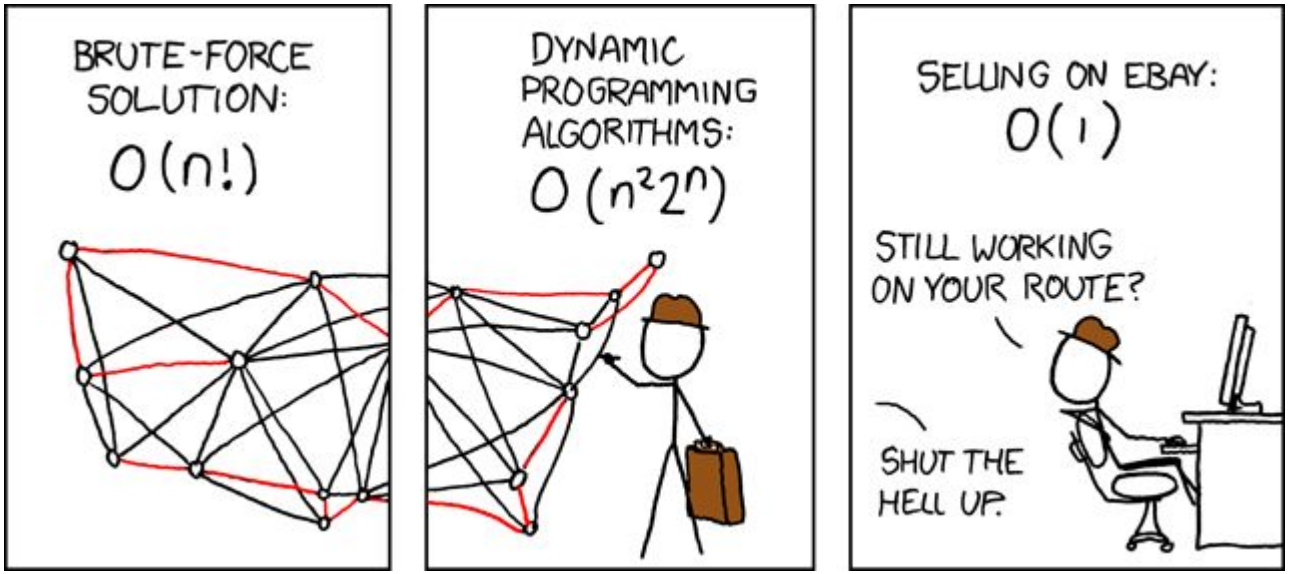
Leistungsverhalten

- **Speicherplatzkomplexität:**
  - Wird primärer & sekundärer Speicherplatz effizient genutzt?
- **Laufzeitkomplexität:**
  - Steht die Laufzeit im akzeptablen / vernünftigen / optimalen Verhältnis zur Aufgabe?
- **Theorie:**
  - liefert untere Schranke, die für jeden Algorithmus gilt, der das Problem löst
- Spezieller Algorithmus liefert obere Schranke für die Lösung des Problems

Laufzeit

Die Laufzeit **T(x)** eines Algorithmus **A** bei Eingabe **x** ist definiert als die Anzahl von Basisoperationen, die Algorithmus **A** zur Berechnung der Lösung bei Eingabe **x** benötigt

Ziel: Laufzeit = Funktion der Größe der Eingabe



Laufzeit

- Sei **P** ein gegebenes Programm und **x** Eingabe für **P**, **|x|** Länge von **x**, und **T(x)** die Laufzeit von **P** auf **x**

- **Ziel:** beschreibe den Aufwand eines Algorithmus als Funktion *der Größe des Inputs*

- **Der beste Fall:**

  $T(n) = inf \{T(x) \mid |x| = n, x \text{ Eingabe für } P\}$

- **Der schlechteste Fall:**

  $T(n) = sup \{T(x) \mid |x| = n, x \text{ Eingabe für } P\}$

Minimum-Suche

Eingabe: Array von `n` Zahlen

Ausgabe: index `i`, so dass `a[i]<a[j]`, für alle `j`

```
def min(A):
    min = 0
    for j in range( 1, len(A) ):
        if A[j] < A[min]:
            min = j
    return min
```

Minimum-Suche

```
def min(A):
    min = 0
    for j in range( 1, len(A) ):
        if A[j] < A[min]:
            min = j
    return min
```

Kosten:	Max Anzahl:
c1	1
c2	n-1
c3	n-1
c4	n-1

Zeit:

$T(n) = c1 + (n-1)\ (c2+c3+c4) < c5n + c1$

n = Größe des Arrays