Informatik I - Übung 7

Pascal Schärli

pascscha@student.ethz.ch

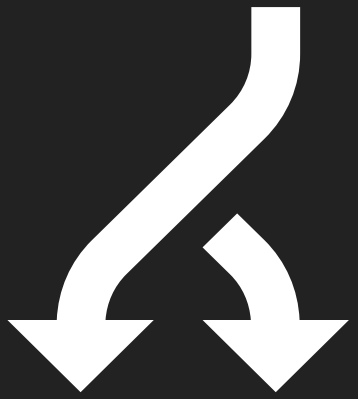
05.04.2019

ETH zürich

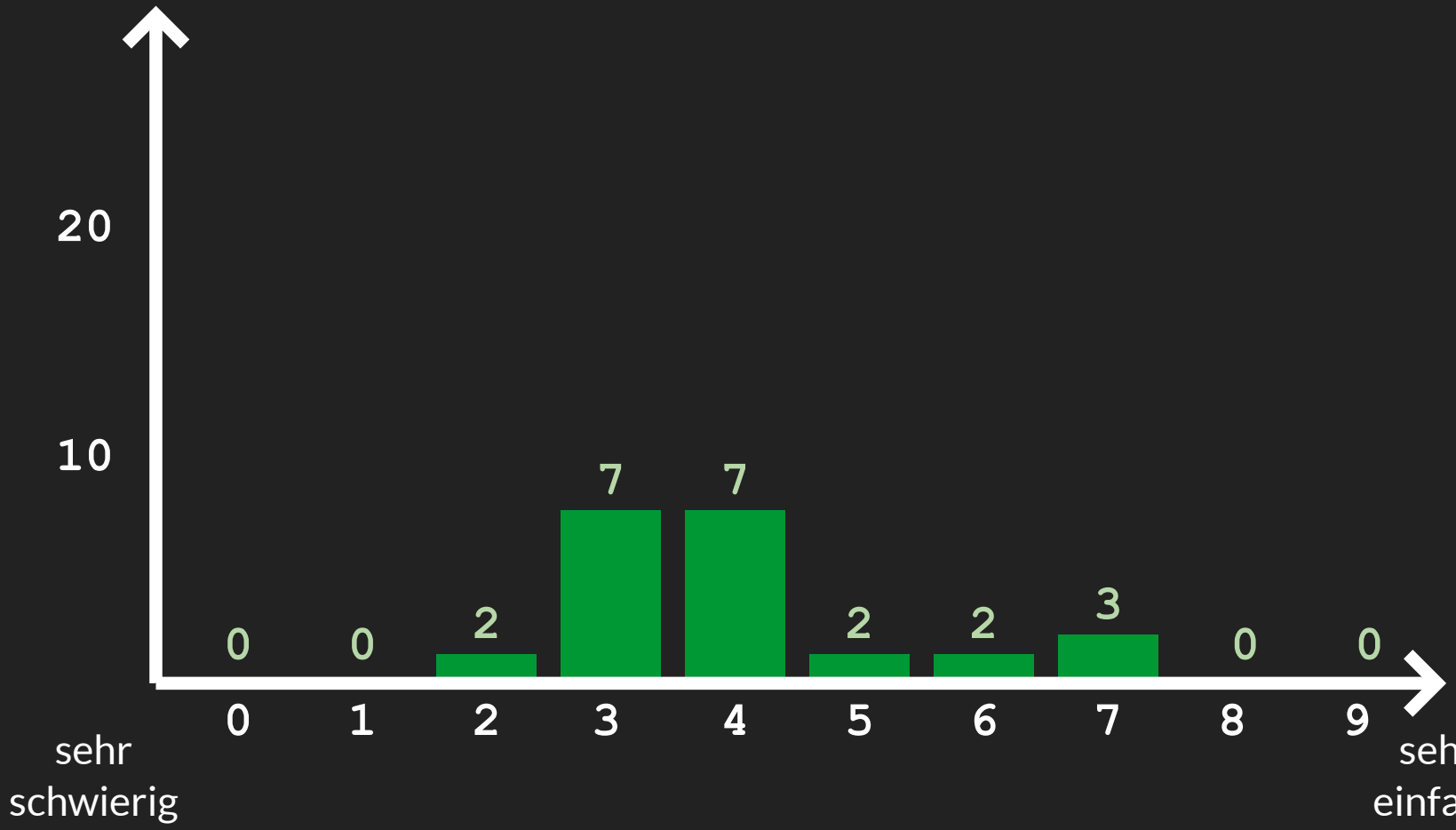
Was gibts heute?

- Feedback
- Best-Of Vorlesung:
  - 2D Vektoren
  - Rekursive Funktionen
- Vorbesprechung
- Program of the Week

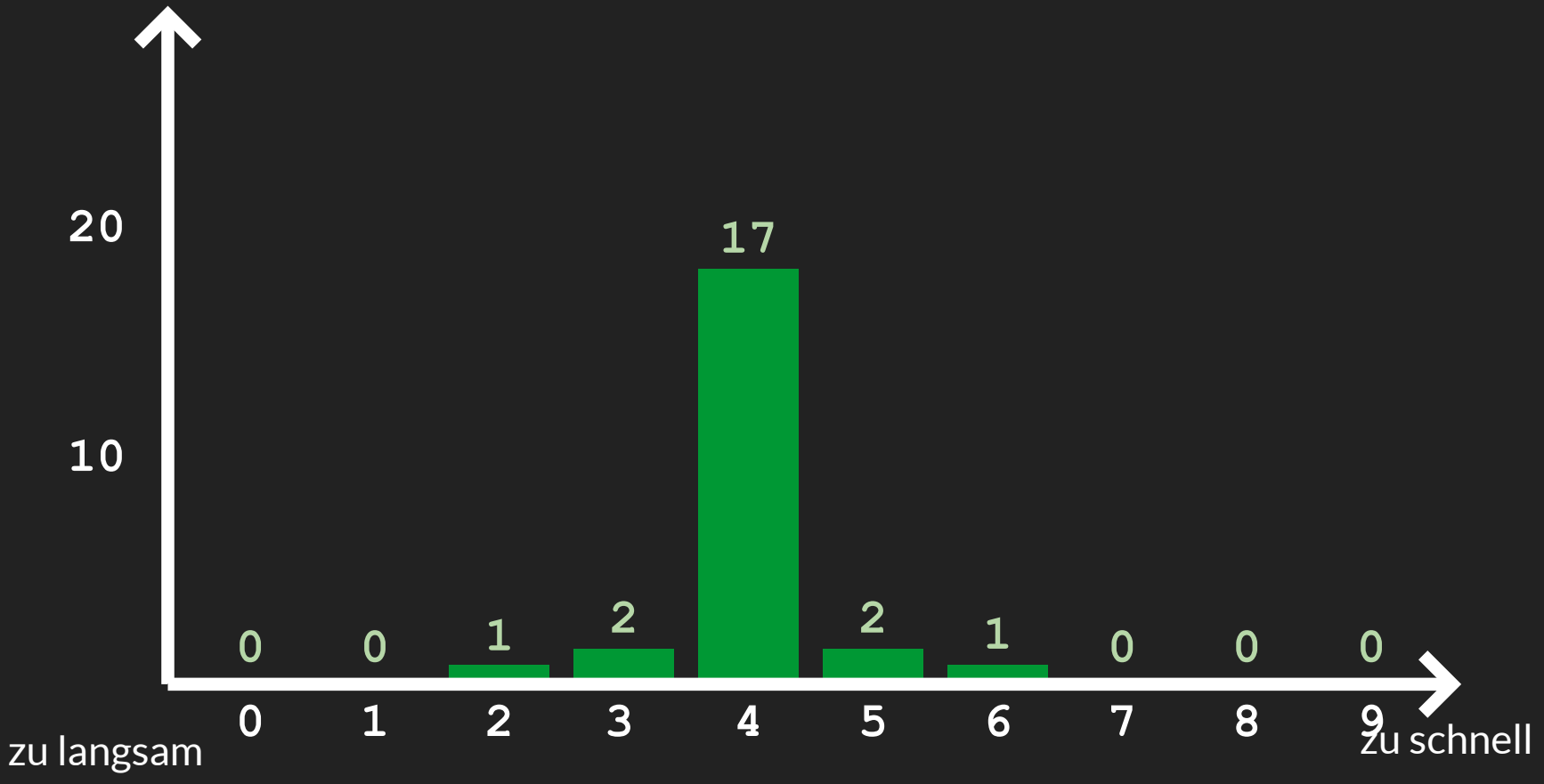
Feedback



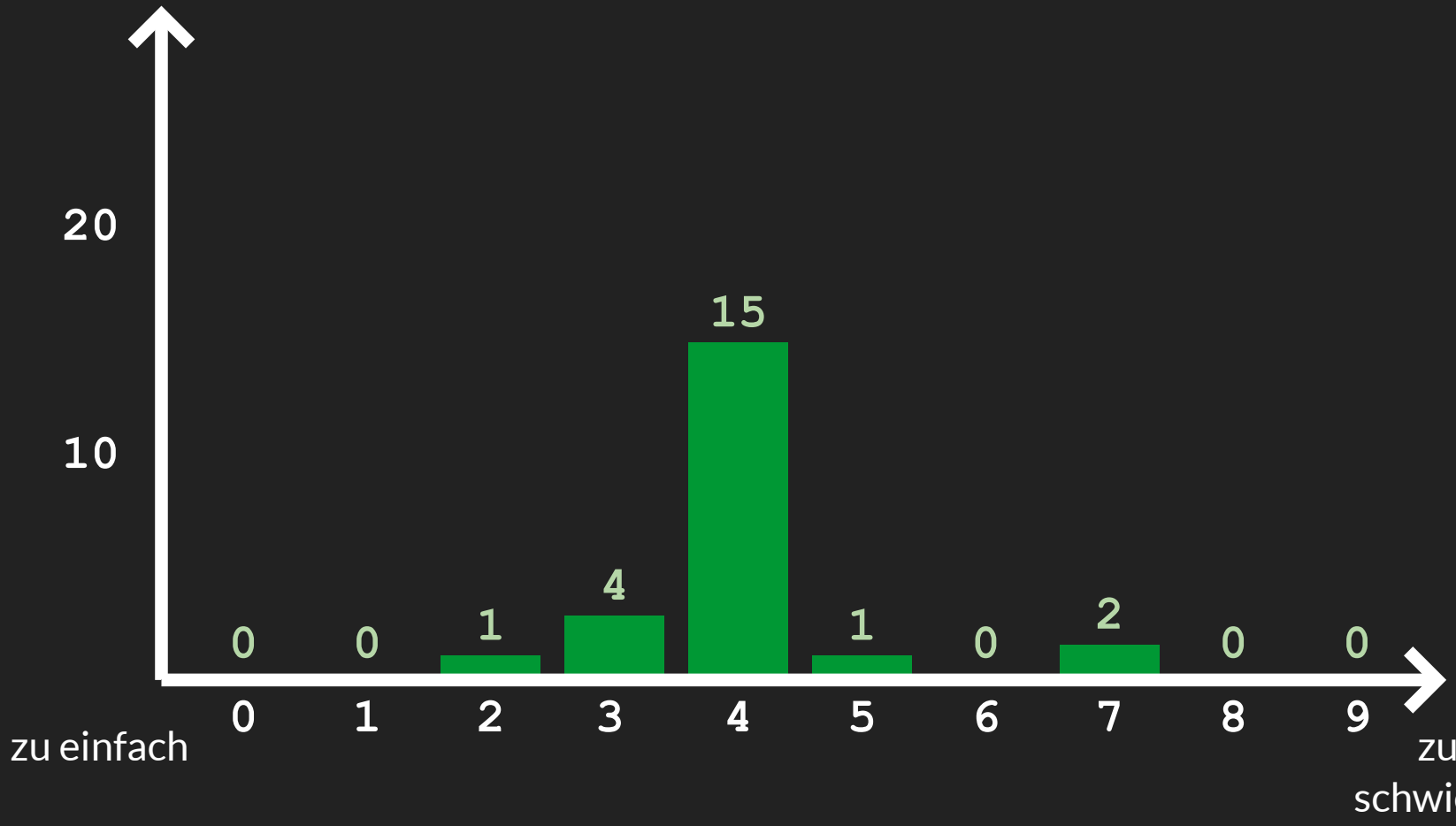
Das Fach Informatik I ist für euch im Allgemeinen:



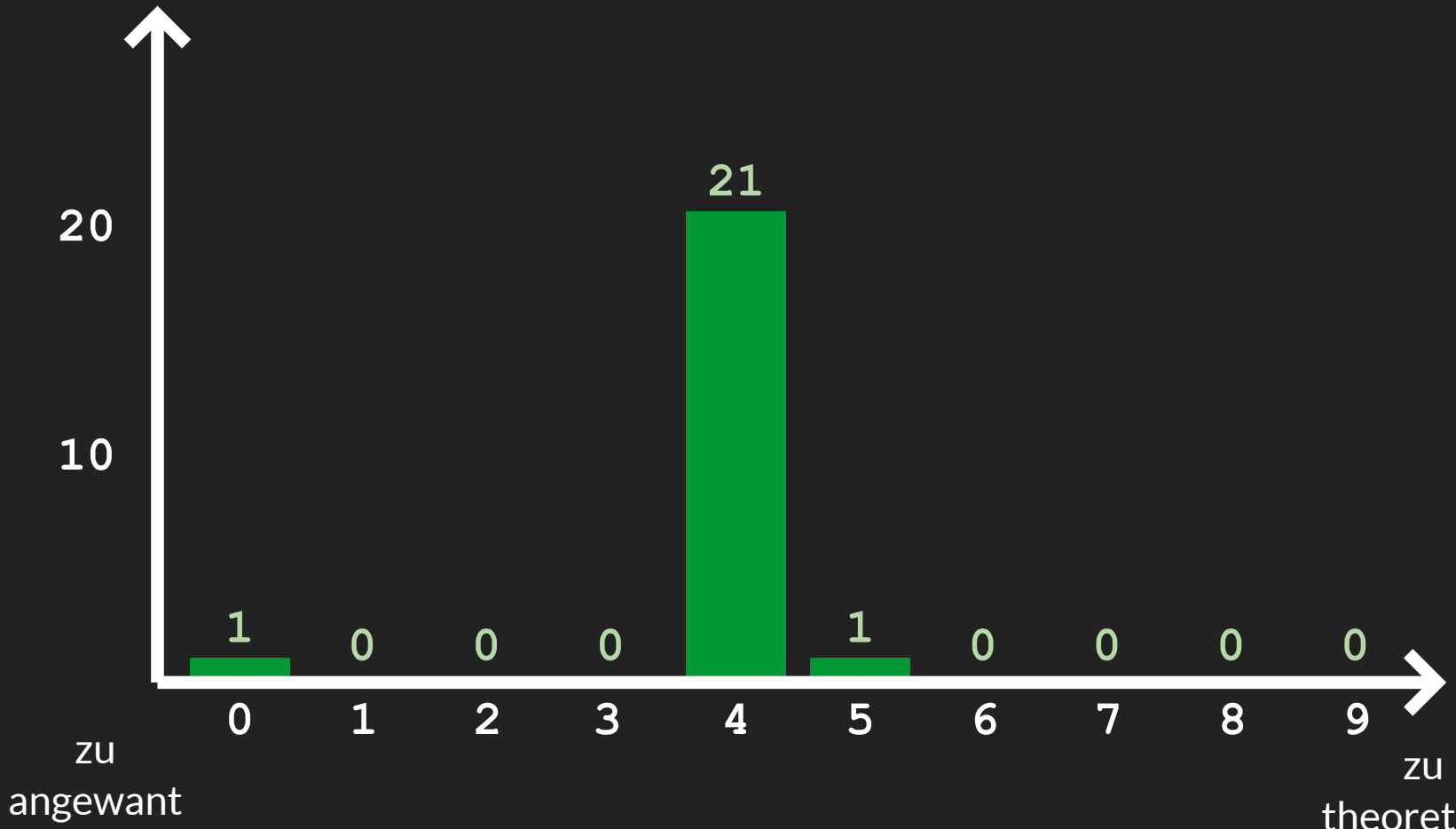
Das Tempo der Übungsstunde ist für euch im Allgemeinen:



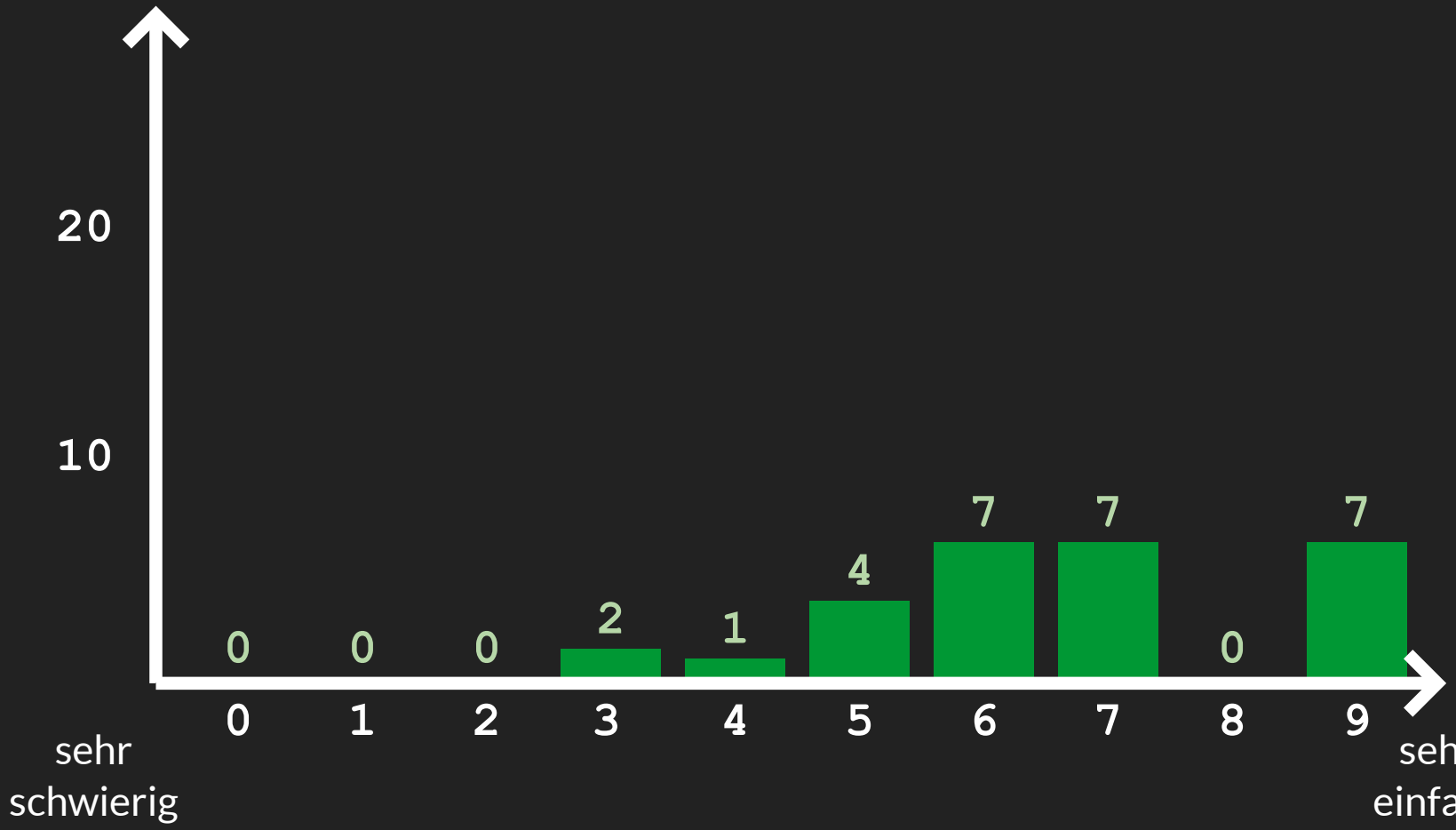
Das Niveau der Übungsstude ist für euch:



Mein Übungsstil ist für euch:



Die Übungsstunden helfen euch so viel beim Lösen der Serien:



Was soll ich ändern?

- 1x Slides vorher hochladen
- 1x Übungsstunde zu "basic"
- 1x Knappere Programme/ Pseudo Code

Was ist gut so?

- 4x Slides
- 4x Program of the Week 
- 3x Beispiele
- 2x Motivation
- 2x Memes
- [...]

Vorlesung

Matrizen in C++

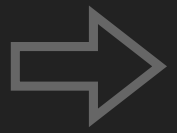
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Wie kann man in C++ 2D Matrizen speichern?

- Ein Vektor kann verschiedene Typen wie int, float oder double speichern.
- Es ist also auch möglich in einem Vector eine Liste von anderen Vektoren zu speichern

Recap

```
std::vector<int> v(3, 0);
```



$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

2-D Vektor

```
std::vector<int> v(3, 0);
```

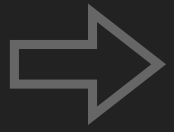
$$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

```
std::vector< std::vector<int> > m(3, std::vector<int>(3, 0));
```

Typ: 1D Vektor

Länge: 3

Startwert: Vektor mit länge 3, startwert 0



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2-D Vektor

```
std::vector< std::vector<int> > vec(3, std::vector<int>(3, 0));
vec[0][0] = 1;
vec[2][1] = 2;
vec[0][2] = 3;
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 3 & 0 & 0 \end{bmatrix}$$

Abkürzungen

```
std::vector< std::vector<int> > m(3, std::vector<int>(3, 0));
```

Damit unser Programm nicht zu unleserlich wird, kann man am Anfang vom Programm (nach den includes) Abkürzungen für Datentypen machen:

```
#include <iostream>
#include <vector>

using vec = std::vector<int>;
using mat = std::vector<vec>;

int main(){
    mat m(3, vec(3, 0));
}
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Abkürzungen

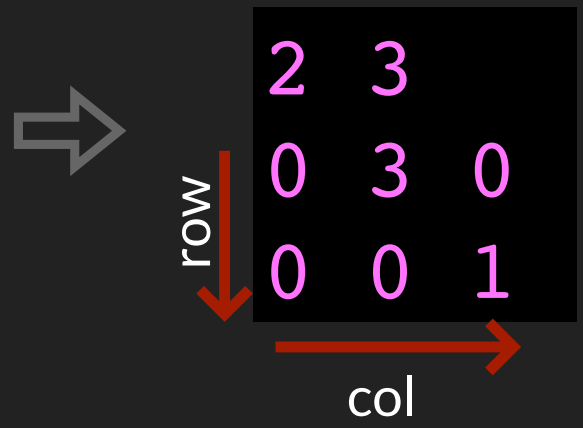
```cpp
#include <iostream>
#include <vector>

using vec = std::vector<int>;
using mat = std::vector<vec>;

int main(){
    vec v(3, 0);
    mat m(2, vec(3, 1));

    m[1][2] = 5;
    v[2] = m[1].size();
    m[1][1] = v.empty();
    v[1] = '9';

    return 0;
}
```

$$v = \begin{bmatrix} 0 & 57 & 3 \end{bmatrix}$$

$$m = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 5 \end{bmatrix}$$

”Printing” Matrices

```
#include <iostream>
#include <vector>
using vec = std::vector<int>;
using mat = std::vector<vec>;

// PRE:  Matrix with at leas one row
// POST: Writes the matrix into standard output.
void print_matrix(const mat &m) {
    unsigned int row, col;
    row = m.size();
    col = m[0].size();

    std::cout << row << " " << col << std::endl;
    for (unsigned int r = 0; r < row; r++) {
        for (unsigned int c = 0; c < col; c++) {
            std::cout << m[r][c] << " ";
        }
        std::cout << std::endl;
    }
}

int main(){
    mat m(2, vec(3,0));

    m[1][2] = 1;
    m[0][1] = 3;

    print_matrix(m);
    return 0;
}
```



Reading Matrices

```
// POST: Returns a matrix that was read from the standard
// input.
mat read_matrix() {
    unsigned int row, col;
    std::cin >> row >> col;
    mat m(row, vec(col));
    for (unsigned int r = 0; r < row; r++) {
        for (unsigned int c = 0; c < col; c++) {
            std::cin >> m[r][c];
        }
    }
    return m;
}
```

Matrix Transpose

Skizziert ein Programm, welches eine Matrix als input nimmt und diese Transponiert:

Input:

```
2 3
1 2 3
4 5 6
```

Output:

```
3 2
1 4
2 5
3 6
```

```
// PRE:  Matrix with at leas one row
// POST: Return a matrix that is a transpose
//       of the input matrix.
mat transpose(const mat &m) {
    unsigned int row = m.size();
    unsigned int col = m[0].size();

    // TODO: Finish implementation.

}
```

```
int main () {
    mat m = read_matrix();
    mat m_transposed = transpose(m);
    print_matrix(m_transposed);
}
```

Matrix Transpose

```
// PRE:  Matrix with at least one row
// POST: Return a matrix that is a transpose
//        of the input matrix.
mat transpose(const mat &m) {
    unsigned int row = m.size();
    unsigned int col = m[0].size();

    mat transposed = mat(col, vec(row));

    for(unsigned int r = 0; r < row; r ++){
        for(unsigned int c = 0; c < col; c++){
            transposed[c][r] = m[r][c];
        }
    }

}
```

Rekursive Funktionen

```
int foo(double p, unsigned int e){
    if(e == 0){
        return 1;
    }
    else{
        return p * foo(p, e-1);
    }
}
```

```
int main(){
    std::cout << foo(2,2) << std::endl;
    std::cout << foo(2,3) << std::endl;
    std::cout << foo(5,2) << std::endl;
    std::cout << foo(10,4) << std::endl;
    return 0;
}
```

```
4
8
25
10000
```

Rekursive Funktionen

```
int foo(double p, unsigned int e){
    if(e == 0){
        return 1;
    }
    else{
        return p * foo(p, e-1);
    }
}
```

$$foo(2,3) \rightarrow p=2, e=3 \rightarrow 2 * foo(2,2) \quad = 2*4=8$$

$$foo(2,2) \rightarrow p=2, e=2 \rightarrow 2 * foo(2,1) \quad = 2*2=4$$

$$foo(2,1) \rightarrow p=2, e=1 \rightarrow 2 * foo(2,0) \quad = 2*1=2$$

$$foo(2,0) \rightarrow p=2, e=0 \rightarrow 1$$

Rekursive Funktionen

```
int foo(unsigned int l, unsigned int b){
    if(l == 1){
        return 0;
    }
    else{
        return 1 + foo(l / b, b);
    }
}
```

```
int main(){
    std::cout << foo(2,2) << std::endl;
    std::cout << foo(5,2) << std::endl;
    std::cout << foo(27,3) << std::endl;
    std::cout << foo(30,3) << std::endl;
    return 0;
}
```

```
1
2
3
3
```

Rekursive Funktionen

```
int foo(unsigned int l, unsigned int b){
    if(l == 1){
        return 0;
    }
    else{
        return 1 + foo(l / b, b);
    }
}
```

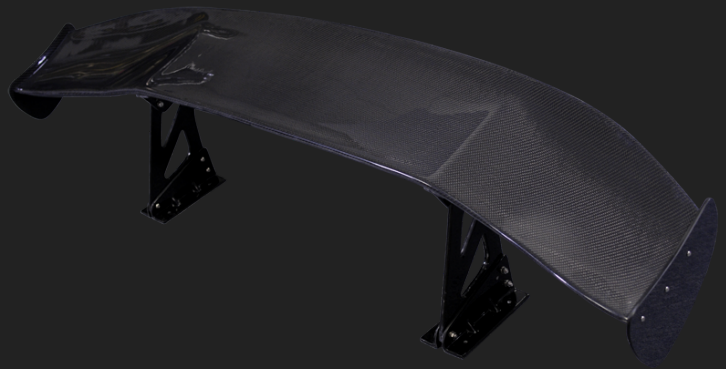
$$foo(30,3) \rightarrow l = 30, b = 3 \rightarrow 1 + foo(10,3) \quad = 1 + 2 = 3$$

$$foo(10,3) \rightarrow l = 10, b = 3 \rightarrow 1 + foo(3,3) \quad = 1 + 1 = 2$$

$$foo(3,3) \rightarrow l = 3, b = 3 \rightarrow 1 + foo(1,3) \quad = 1 + 0 = 1$$

$$foo(1,3) \rightarrow l = 1, b = 3 \rightarrow 0$$

Vorbesprechung



Pascal Schärli *05.04.2019*

ETH zürich

1: Vector & Matrix

Schreibt ein Programm, welches Integer Vektoren und Matrizen miteinander multiplizieren kann.

Text Representationen:

Skalar

```
s 7
```

Vektor

```
v 3
3 5 7
```

Matrix

```
m 2 3
  1   1 -5
 -1   0  4
```

1: Vector & Matrix

Einlesen:

Lest zuerst den Charakter ein, welcher den Datentyp bestimmt (s, v oder m). Danach ruft ihr die entsprechende Funkion auf (Matrix einlesen)

Output for X * Y :

X \ Y		**Vektor(ll)**	**Matrix(m2xn2)**
	Vektor(l)	*Skalar*	*error*
	Matrix(m1xn1)	n1 = l, Vektor(m1)	n1 = m2, Matrix(m1xn2)

2: Binary NZZ

Am 8.06.2012 war die Titelseite der NZZ in binär geschrieben. Wir werden nun diese Titelseite als Übung lesen!

01001110 01011010 01011010

⇩

78 90 90

⇩ ASCII

NZZ

2: Binary NZZ

Euer Input kommt aus einem File. Um den Inhalt von diesem File zu bekommen könnt ihr einen sogenannten "Filestream" benutzen:

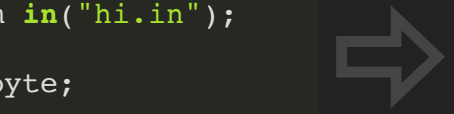
hi.in

```
1101000 1101001 100001
```

+

main.cpp

```cpp
#include <iostream>
#include <string>
#include <fstream>

int main(){
    std::ifstream in("hi.in");

    std::string byte;

    while((bool)(in >> byte)) {
        std::cout << byte << " ";
    }
}
```



```
1101000 1101001 100001
```

2: Binary NZZ

Um einzelne "char" aus eurem byte herauszulesen könnt ihr einen Loop machen und die einzelnen Charakter mit [i] herauslesen.

```
1 std::string string = "01010000";
2
3 for(unsigned int i = 0; i < 8; i++){
4     std::cout << string[i] << '\n';
5 }
```

"double" quotes für string

'single' quote für char

Achtung! Diese sind vom typ "char", und nicht "int".

'0' -> 48

'1' -> 49

Siehe Slides vom letzten mal!

2: Binary NZZ

Starthilfe (Nur für den Notfall)

```cpp
#include <iostream>
#include <string>
#include <fstream>

// POST: Returns the character corresponding to the
//        input string value
char decode_to_char(const std::string & value){
    // TODO
}

int main () {
    // Read file name and open file input stream.
    std::string filename;
    std::cin >> filename;
    std::ifstream in(filename);
    if(!(bool)(in)) {
        std::cout << "File does not exists !\n";
        return 1;
    }

    std::string value;
    while((bool)(in >> value)) {
        std::cout << decode_to_char(value);
    }

    return 0;
}
```

3: Recursive Functions

```
bool f(const int n) {
  if (n == 0){
    return false;
  }
  else{
    return !f(n-1);
  }
}
```

```
void g(const int n) {
  if (n == 0) {
    std::cout << "*";
    return;
  }
  g(n-1);
  g(n-1);
}
```

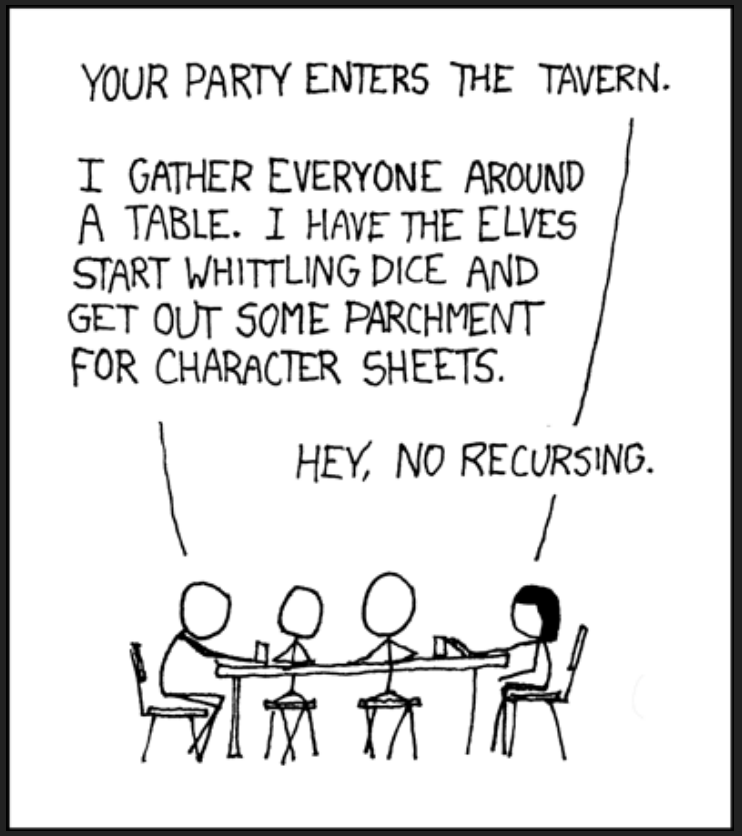
1. Pre-/Post Conditions
2. Begründe warum die Programme terminieren
3. Berechnet die Anzahl Funktionsaufrufe in Abhängigkeit von n

Tipps:

- Überlegt euch ein Beispiel falls ihr nicht weiter kommt.
- Eventuell hilft es die Funktion in einem Programm zu testen

Program of the Week

Viel Spass!


YOUR PARTY ENTERS THE TAVERN.
I GATHER EVERYONE AROUND A TABLE. I HAVE THE ELVES START WHITTLING DICE AND GET OUT SOME PARCHMENT FOR CHARACTER SHEETS.
HEY, NO RECURSING.