Rust-Merkblatt

```
use std::mem::replace;
replace<T>(dest: &mut T, src: T) -> T

use std::io;
io::stdin().read_line(&mut input)?

use std::io::Write;
io::stdout().flush()?;
f.write_all(s.as_bytes())?;

use std::fs::File;
let mut f = File::open("a.txt")?;
let mut f = File::create("a.txt")?;

use std::io::Read;
f.read_to_string(&mut s)?;
f.read_to_end(&mut bv)?;

let v: Vec<T> = Vec::new();
let v: Vec<T> = Vec::with_capacity(n);
let v = vec![1,2,3,4];
let v = vec![0;n];
let a = v.into_boxed_slice();
v.push(x);
let n = v.len();
let n = v.capacity();
let condition = v.is_empty();
v.reserve(n);
v.append(&mut v2);
v.clear();

a.swap(i,j);
a.reverse();
let i = a.iter();
let i = a.iter_mut();
a.sort();
a.sort_by_key(|x| x.abs());

let m = HashMap::new();
m.insert(key,value);
let condition = m.contains_key(key);
m.remove(key);
let value = m.get(key)?; (Option)
for (key,value) in &m {}
#[derive(Hash, Eq, PartialEq, Debug)]
struct P{x: u32, y: u32}
let m: HashMap<P,T> = HashMap::new();
```

```
use std::rc::Rc;
use std::cell::RefCell;
use std::cell::Cell;
use std::collections::HashMap;
use std::fmt;

i.all(|&x| x%2==0)
i.any(|&x| x%2==0)
i.filter(|x| x%2==0)
i.map(|x| 2*x)
i.count()
i.nth(n)
i.chain(j)
i.zip(j)
i.enumerate()
i.for_each(f)

use std::mem;
let n = mem::size_of::<u32>();
let n = mem::size_of_val(&x);
```

```
impl fmt::Display for T {
  fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
    write!(f, "{}", ...)
  }
}
```