* Immutability by default

[1]

```
let a = 5;
a = 3; // error[E0384]: re-assignment of immutable variable `a`
```

[2]

```
let s = String::from("hello");
s.push_str(" world"); //error[E0596]: cannot borrow immutable local variable `s` as mutable
```

* Nullptr dereference situation

```
struct Entity {}
fn inspect(e: Entity) {}

let e: Entity;
inspect(e); // error[E0381]: use of possibly uninitialized variable: `e`
```

* RAII Idiom

```
{
    let s = String::from("hello"); // s is valid from this point forward
    // do stuff with s
}  // this scope is now over, and s is no longer valid
```

https://stackoverflow.com/questions/2321511/what-is-meant-by-resource-acquisition-is-initialization-raii
https://en.wikibooks.org/wiki/More_C%2B%2B_Idioms/Resource_Acquisition_Is_Initialization

* Move semantics (Ownership – only one owner at a time)

[1]

```
let s1 = String::from("hello");
let s2 = s1;
println!("{}, world!", s1); // error[E0382]: use of moved value: `s1`
```

[2]

```
fn add_first_three(v: Vec<i32>) -> i32 {
    v[0] + v[1] + v[3]
}

let v = vec![1, 2, 3, 4, 5];
let sum_of_first_three= add_first_three(v);
println!("{} + {} + {} = {}", v[0], v[1], v[2], sum_of_first_three); // error[E0382]: use of moved value: `v`
```

https://mbevin.wordpress.com/2012/11/20/move-semantics/
https://www.cprogramming.com/c++11/rvalue-references-and-move-semantics-in-c++11.html

* Borrowing

[1]

```
fn add_first_three_without_moving(v: &Vec<i32>) -> i32 {
    v[0] + v[1] + v[3]
}
```

[2]

```
let mut s = String::from("hello world");
let immutable_reference1 = &s;
let immutable_reference2 = &s;
let mutable_reference = &mut s; // error[E0502]: cannot borrow `s` as mutable because it is also borrowed as immutable
```

[3]

```
let mut s = String::from("hello world");
let mutable_reference1 = &mut s;
let mutable_reference2 = &mut s; // error[E0499]: cannot borrow `s` as mutable more than once at a time
```

We can have:
- **many immutable references** to a resource
- only **one mutable reference** to a resource

but only ONE of them at the same time!

(+ Borrower's scope must not outlast the owner)

Btw:

```
fn dangle() -> &String {
    let s = String::from("hello world");
    &s // error[E0106]: missing lifetime specifier
}
```

* Monadic types / Exception handling

Option<T>

```
fn divide(numerator: f64, denominator: f64) -> Option<f64> {
    if denominator == 0.0 {
        None
    } else {
        Some(numerator / denominator)
    }
}
match divide(2.0, 3.0) {
    Some(x) => println!("Result: {}", x),
    None    => println!("Cannot divide by 0"),
}
```

https://doc.rust-lang.org/std/option/enum.Option.html

Result<T>

```
fn cat(path: &Path) -> io::Result<String> {
    let mut f = File::open(path)?;
    let mut s = String::new();
    match f.read_to_string(&mut s) {
        Ok(_) => Ok(s),
        Err(e) => Err(e),
    }
}
```

https://doc.rust-lang.org/std/result/enum.Result.html

Either<L, R>

https://rust-bio.github.io/rust-bio/either/enum.Either.html

Future<Item, Error>

```
fn download(url: &str) -> Box<Future<Item=File, Error=io::Error>> {
    let data = resolve(url)
        .and_then(|addr| connect(&addr))
        .and_then(|conn| download(conn))
        .map(|data| parse::<File>(data));

    Box::new(data)
}
```

https://theta.eu.org/2017/08/04/async-rust.html
https://tokio.rs/docs/getting-started/futures/
https://aturon.github.io/blog/2016/08/11/futures/

* Functional constructs

```
let plus_one = |x:u32| -> u32 { x + 1 };

let sum_of_first_hundred_odd_numbers =
        (0..)
        .map(|x| x * x)
        .take_while(|&x| x <= 100)
        .filter(|x| x % 2 == 1)
        .fold(0, |sum, x| sum + x);
```

http://science.raphael.poss.name/rust-for-functional-programmers.html
http://xion.io/post/programming/rust-into-haskell.html

* Pattern matching

[1]

```
fn how_many(x:i32) -> &'static str {
    match x {
        0 => "no",
        1 | 2 => "one or two",
        12 => "a dozen",
        9...11 => "lots of",
        _ if (x % 2 == 0) => "some",
        _ => "a few"
    }
}
```

[2]

```
enum OptionalInt {
    Value(i32),
    Missing,
}

let x = OptionalInt::Value(5);
match x {
    OptionalInt::Value(i) if i > 5 => println!("Got an int bigger than five!"),
    OptionalInt::Value(..) => println!("Got an int!"),
    OptionalInt::Missing => println!("No such luck."),
}
```

[3]

```
struct Point {
    x: i32,
    y: i32,
}
let origin = Point { x: 0, y: 0 };
match origin {
    Point { y, .. } => println!("y is {}", y),
}
```

* FFI (Foreign Function Interface)

```
use libc::size_t;
extern {
    fn function_in_c(len: size_t) -> size_t;
}
```

http://siciarz.net/24-days-of-rust-calling-rust-from-other-languages/
https://github.com/alexcrichton/rust-ffi-examples

* Multithreading, Smart pointers, Generics, Traits, Macros

;(

Rust is loved!!!
https://insights.stackoverflow.com/survey/2016
https://insights.stackoverflow.com/survey/2017

Rust performance:
vs C - https://benchmarksgame.alioth.debian.org/u64q/rust.html
vs C++ - https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=rust&lang2=gpp
vs Go - https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=rust&lang2=go
vs Java - https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=rust&lang2=java

Rust Documentation - https://doc.rust-lang.org/
Rust FAQ - https://www.rust-lang.org/en-US/faq.html
Rust Book - https://doc.rust-lang.org/book/
Rust by Example - https://rustbyexample.com/index.html
Rust on YT - https://www.youtube.com/channel/UCaYhcUwRBNscFNUKTjgPFiA

Servo Engine - https://servo.org/

Redox - https://www.redox-os.org/
Re-implementing linux syscalls in Rust - https://dominuscarnufex.github.io/cours/rs-kernel/en.html
Writing an OS in Rust - https://os.phil-opp.com/
“Tifflin” Experimental Kernel in Rust - https://github.com/thepowersgang/rust_os