RUST systems programming language

Rust is a systems programming language that runs blazingly fast, prevents almost all crashes*, and eliminates data races.

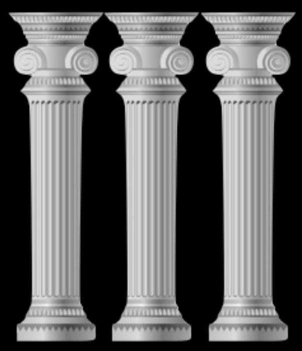
There is no garbage collection in Rust. The lifetime of a variable and de-allocates it where it is no longer used.

Super fast code generation.

C function compatibility (extern C).

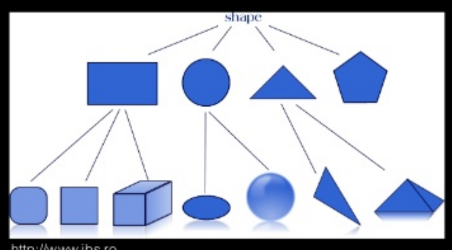
Simpler syntax than C++.

Rust Pillars



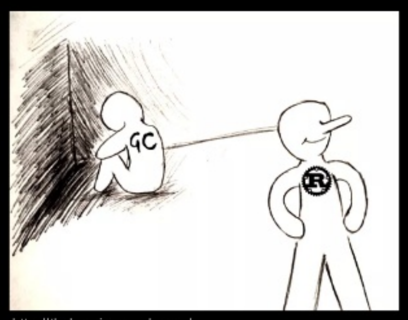
- Memory safety without GC (garbage collector)
- Abstraction without overhead.
- Concurrency without data races.

Abstraction Without Overhead



http://www.ibs.ro

Memory Safety Without GC



http://theburningmonk.com/

Concurrency Without Data Races

- Send trait
- Sync trait

http://en.wikipedia.org

primatives

```
fn main() {
    //integers
    let i: i8 = 1; // i16, i32, i64, and i are available
    //unsigned
    let u: u8 = 2; // u16, u32, u64, and u are available
    //floats
    let f: f32 = 1.0; // f64 also available
    //booleans
    let b: bool = true; // false also available, duh
    //string and characters
    let c: char = 'a';
    let s: &str = "hello world";
}
```

variable bindings

```
fn main() {

    let x: int = 1;                 //explicitly declare type
    let y = 2i;                     //type inference
    let (a,b,c) = (1i,2i,3i);       //variable declaration via patterns
    let a = [1, 2, 3];              //array literals
    let s = "hello";                //string literal

  println!("*_^ x = {}, y = {}, a,b,c = {},{},{}", x,y,a,b,c);
}

//> Cargo run
//> *_^ x = 1, y = 2, a,b,c = 1,2,3
//
```

variable mutability

```
fn main() {

    let x: int = 1;
    x = 10;

    println!("The value of x is {}", x);
}

//Cargo run
//error: re-assignment of immutable variable `x`
//     x = 10;
//     ^~~~~~~
```

stack vs. heap

```
fn main() {

    let y: int = 1;                  //allocated on the stack
    let x: Box<int> = Box::new(10); //allocated on the heap

    println!("Heap {}, Stack {}", x, y);
}

//> Cargo run
//> Heap 10, Stack 1
//
```

Other Features

- Pattern Matching
- Enums
- Closures
- Macro

http://www.viswasenterprises.in

memory mutability

```
fn main() {

    let x: Box<int> = box 10;   //allocated on the heap
    *x = 11;

    println!("The Heaps new value is {}, x");
}

//Cargo run
//error: cannot assign to immutable dereference of `*x`
//     x = 11;
//     ^~~~~~~
```

rust

1. guaranteed memory safety
2. threads without dataraces
3. zero-cost abstractions (done at compile time)
4. trait-based generics
5. pattern matching
6. type inference
7. & more

Conclusion

Gives control without compromising safety

Zero cost abstractions

Zero cost safety

Guarantees beyond dangling pointers

Iterator invalidation in a broader sense