Rust i Sztuczna Inteligencja

Paweł Czapiewski 09.03.2023

Paweł Czapiewski

- Graduated:

Karlsruher Institut für Technologie

POLITECHNIKA GDAŃSKA

- Programming Langueas:

- Working in

Passion for Technology

PJATK 学大科工報情本日ポーランド

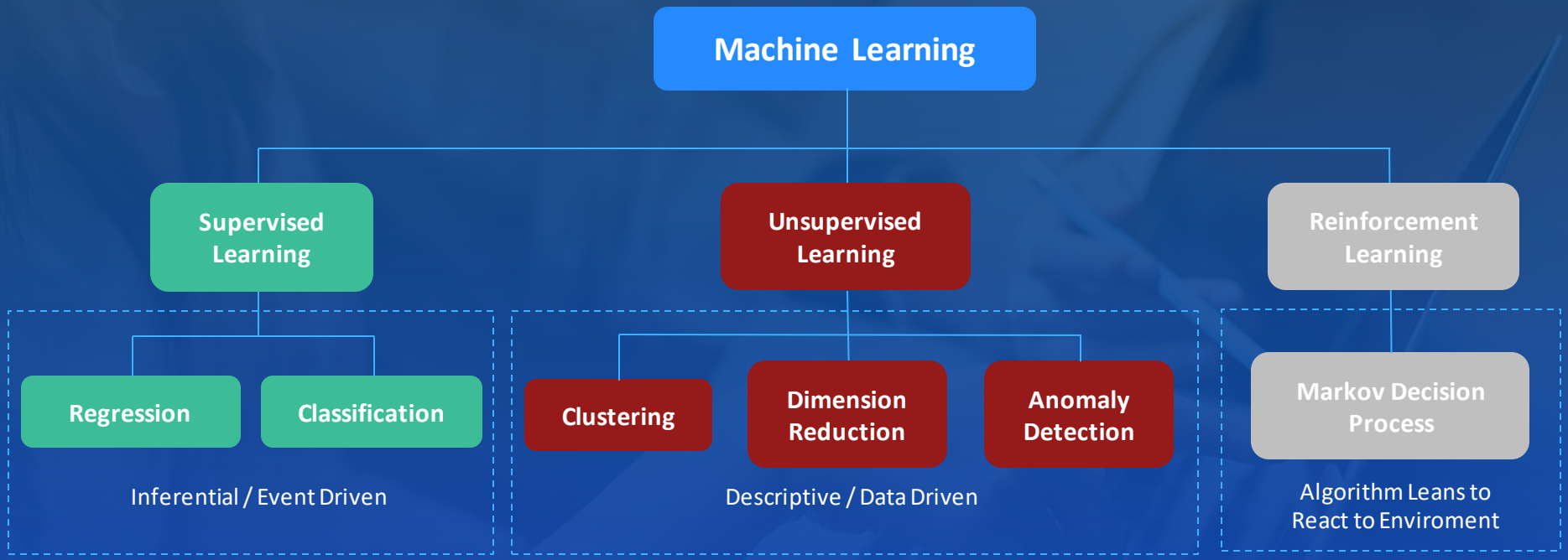
- Hobby:

AI

Agenda

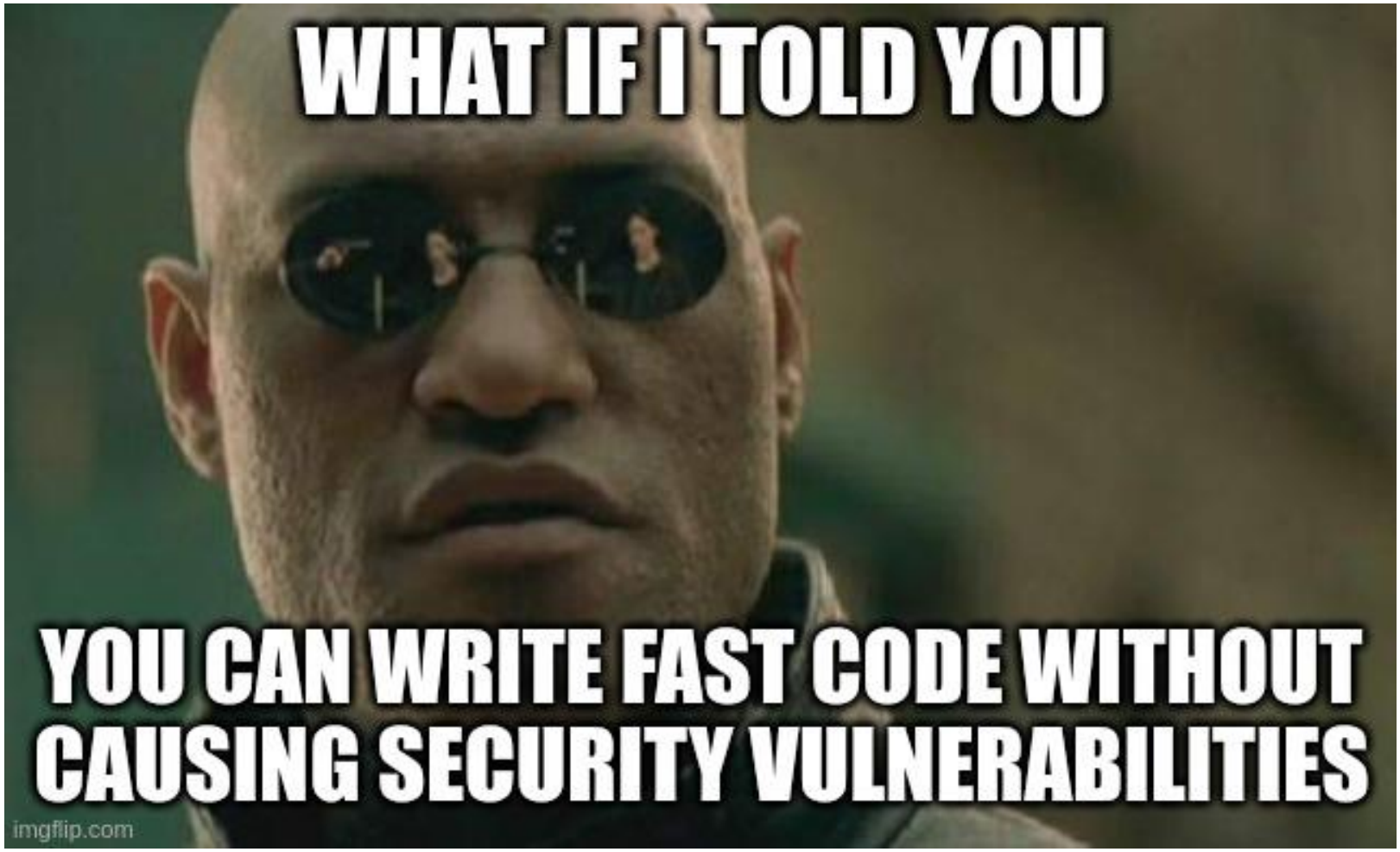
1. Rust Concept
2. Rush Crash Curse
3. Example Implementation of the Unsupervised ML
4. Example Implementation of Supervised ML
5. Example Implementation of the Reinforcement Learning

Machine Learning



Source: Mastering Machine Learning with Python in Six Steps

Title

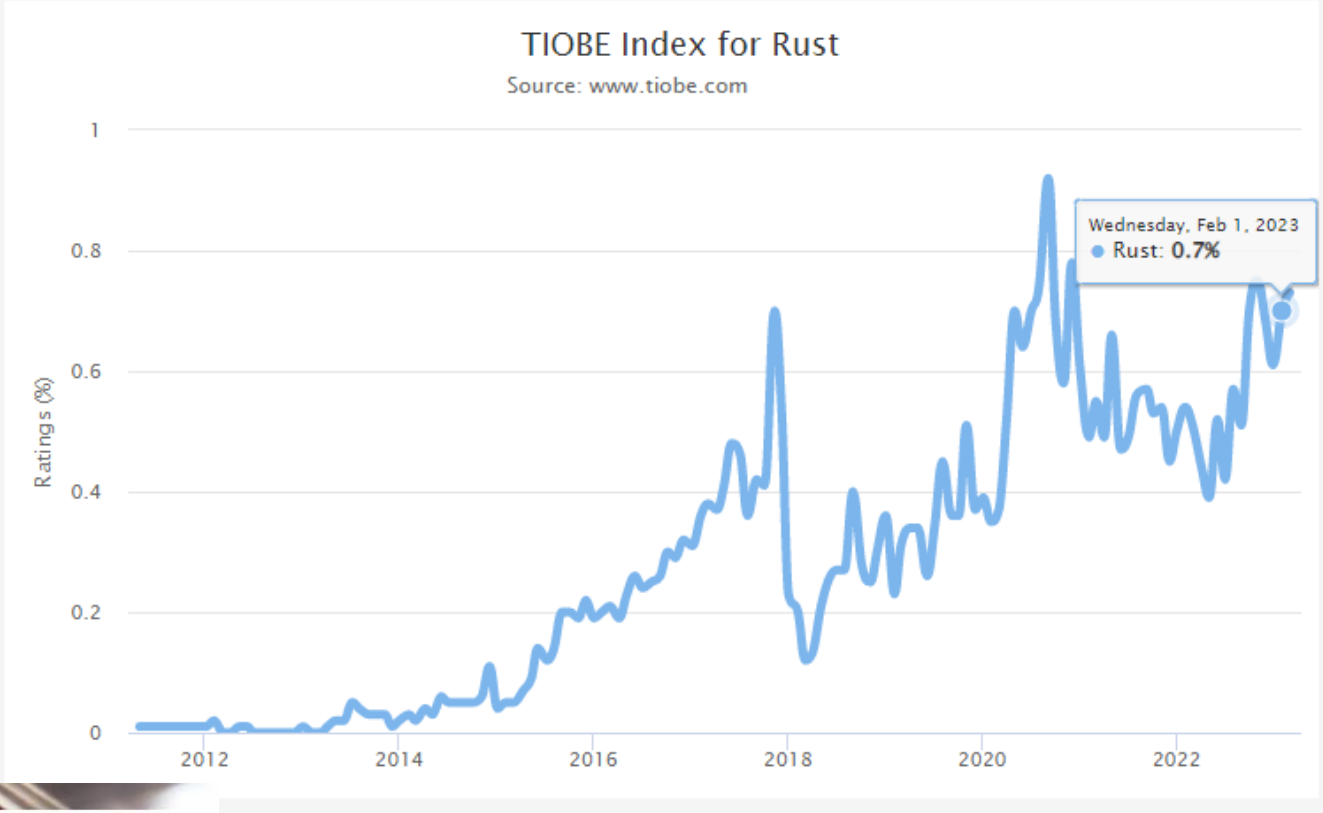


Source: https://developer.okta.com/blog/2022/03/18/programming-security-and-why-rust

RUST Popularity

Highest Position (since 2011): #18 in Dec 2022

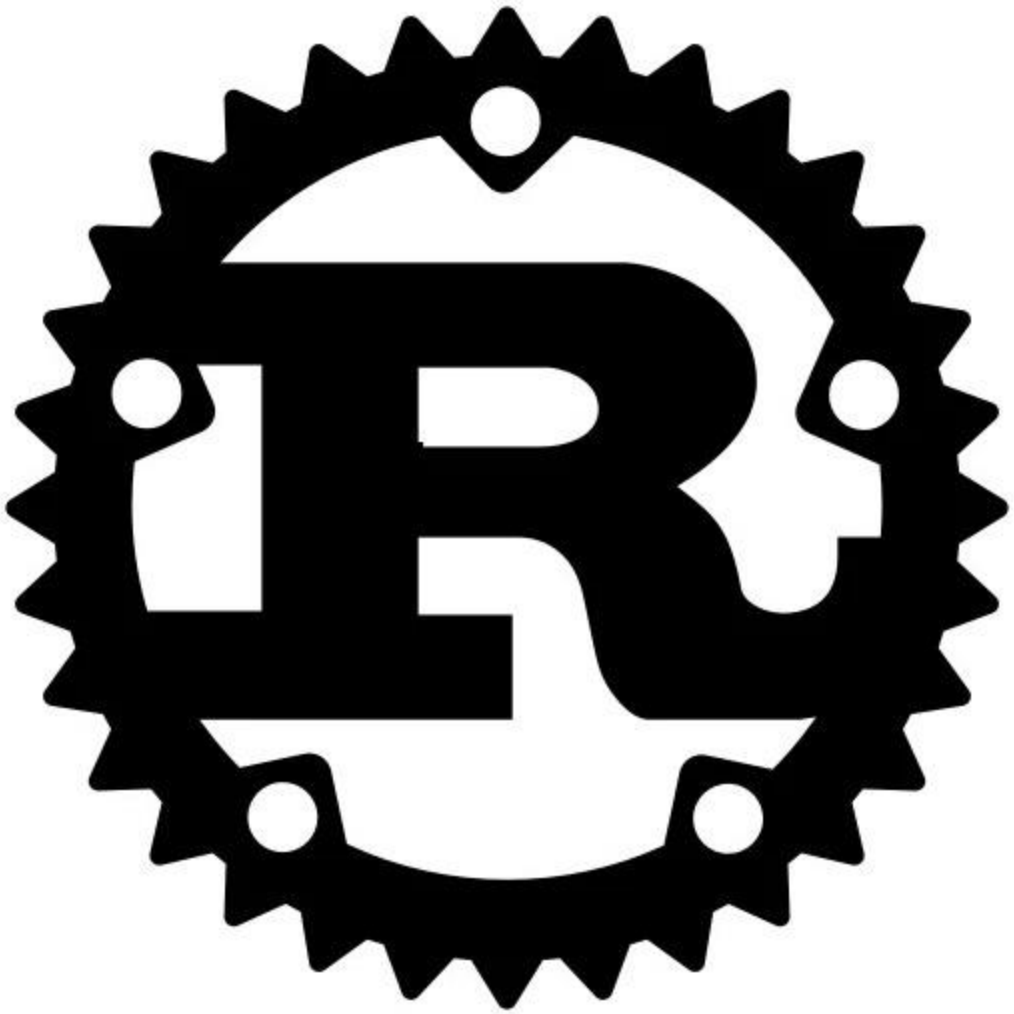
Lowest Position (since 2011): #211 in Dec 2012



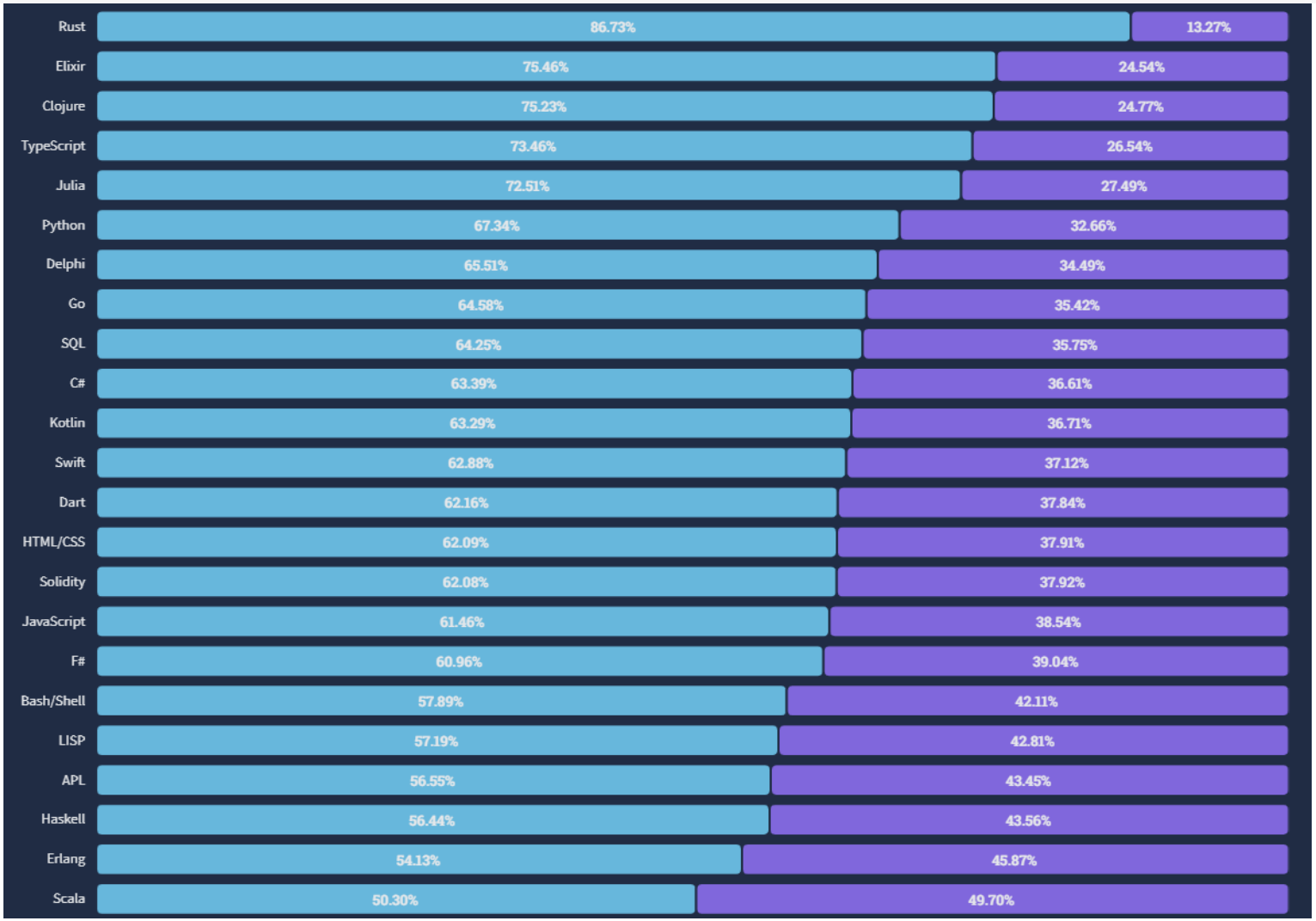
TIOBE
the software quality company

Source: https://www.tiobe.com/tiobe-index/rust/

RUST Popularity

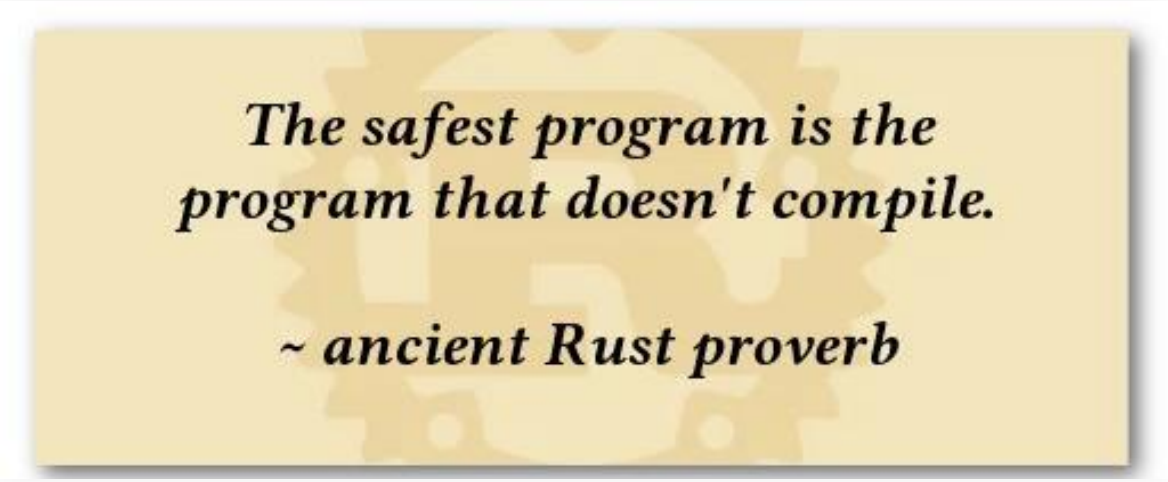


Stack Overflow Developer Survey – loved vs Dread



Source: Stack Overflow Developer Survey 2022

RUST Popularity



Source: https://raw.githubusercontent.com/rochacbruno/rust_memes/master/img/safest_code.jpg

RUST Popularity

> "Rust, not Firefox, is Mozilla's greatest industry contribution."
>
> – TechRepublic

Source: https://developer.okta.com/blog/2022/03/18/programming-security-and-why-rust

Source: https://www.devopsschool.com/blog/top-50-rust-programming-interview-questions-and-answers/

Rustacean

Search for a Rustacean: search

(by name, irc nick, username for Reddit, GitHub, Discourse, etc.)

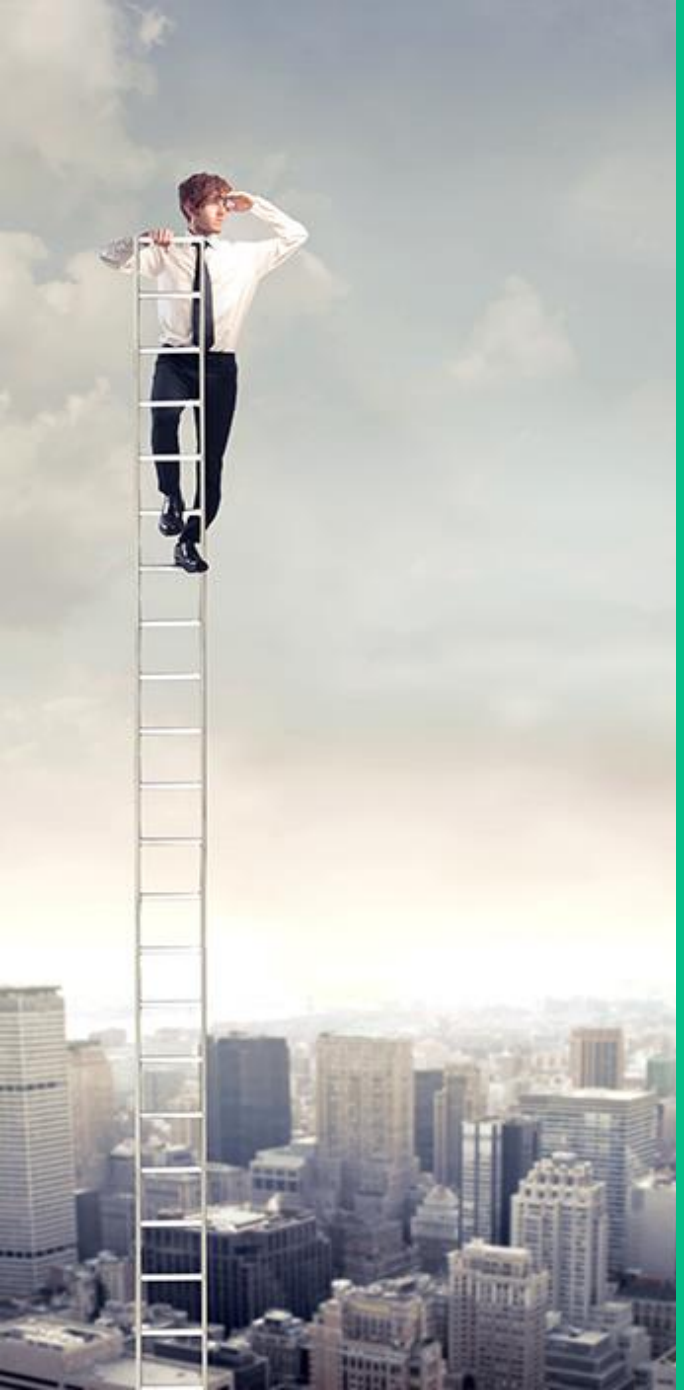
Rustaceans communicate via many channels:

- Discourse (users): for discussing using and learning Rust.
- Discourse (internals): for discussion of Rust language design and implementation. And bike-shedding.
- Reddit: for general Rust discussion.
- IRC on Moznet:
  - #rust is for all things Rust;
  - #rust-internals is for discussion of other Rust implementation topics;
  - #rustc is for discussion of the implementation of the Rust compiler;
  - #rust-lang is for discussion of the design of the Rust language;
  - #rust-libs is for discussion of the implementation of the Rust standard libraries;
  - #rust-tools is for discussion of Rust tools;
  - #rust-gamedev is for people doing game development in Rust;
  - #rust-crypto is for discussion of cryptography in Rust;
  - #rust-osdev is for people doing OS development in Rust;
  - #rust-webdev is for people doing web development in Rust;
  - #rust-networking is for people doing computer network development and programming in Rust;
  - #cargo is for discussion of Cargo, Rust's package manager;
  - #rust-offtopic is for general chit-chat amongst Rustaceans;
  - #servo is for discussion of Servo, the browser engine written in Rust;
  - #rust-bots notifcations about Rust from a selection of bots.

Source: https://www.rustaceans.org/

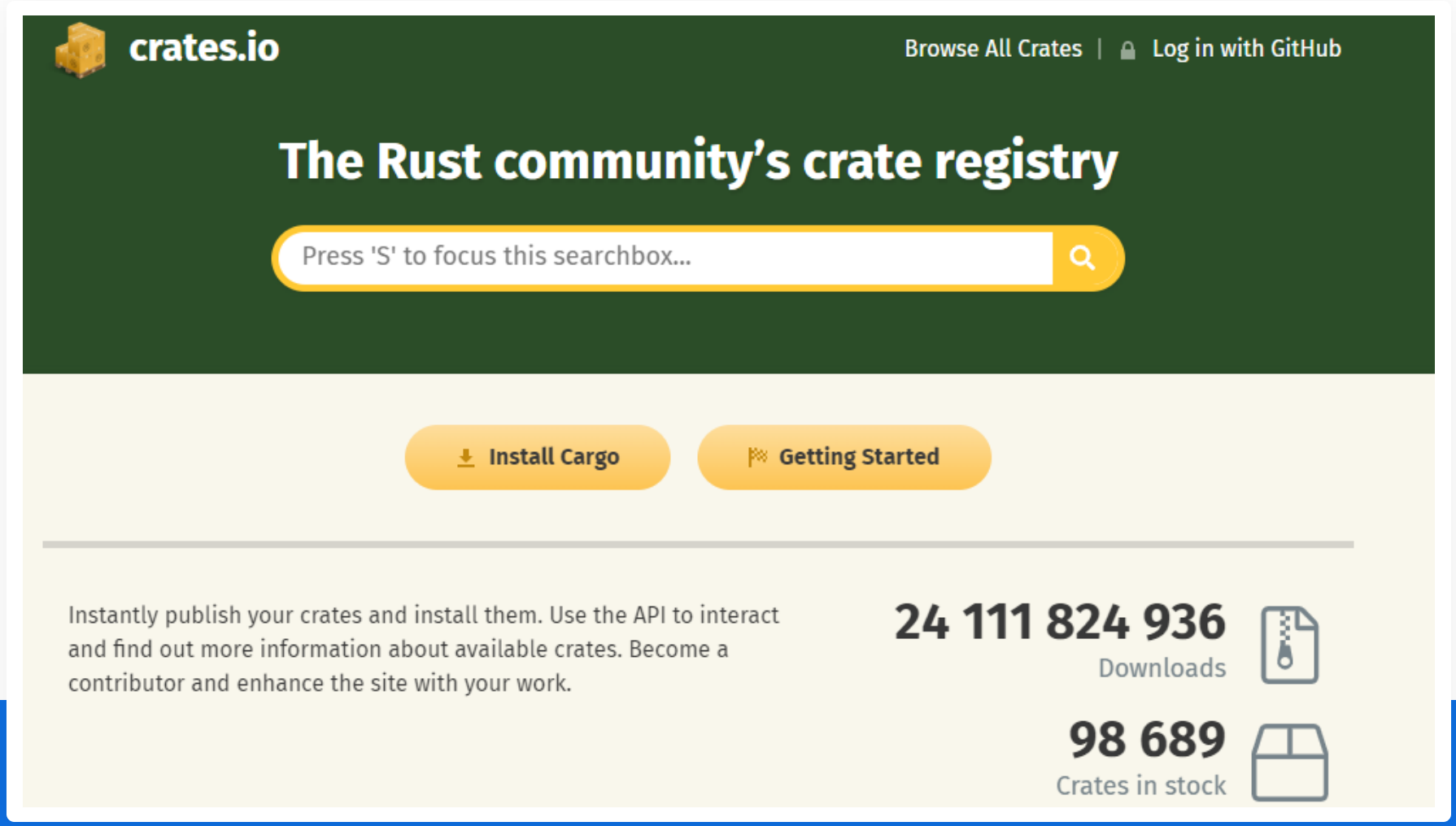
Rustacean

Source: https://github.com/rochacbruno/rust_memes/blob/master/img/python_for_kids.jpg

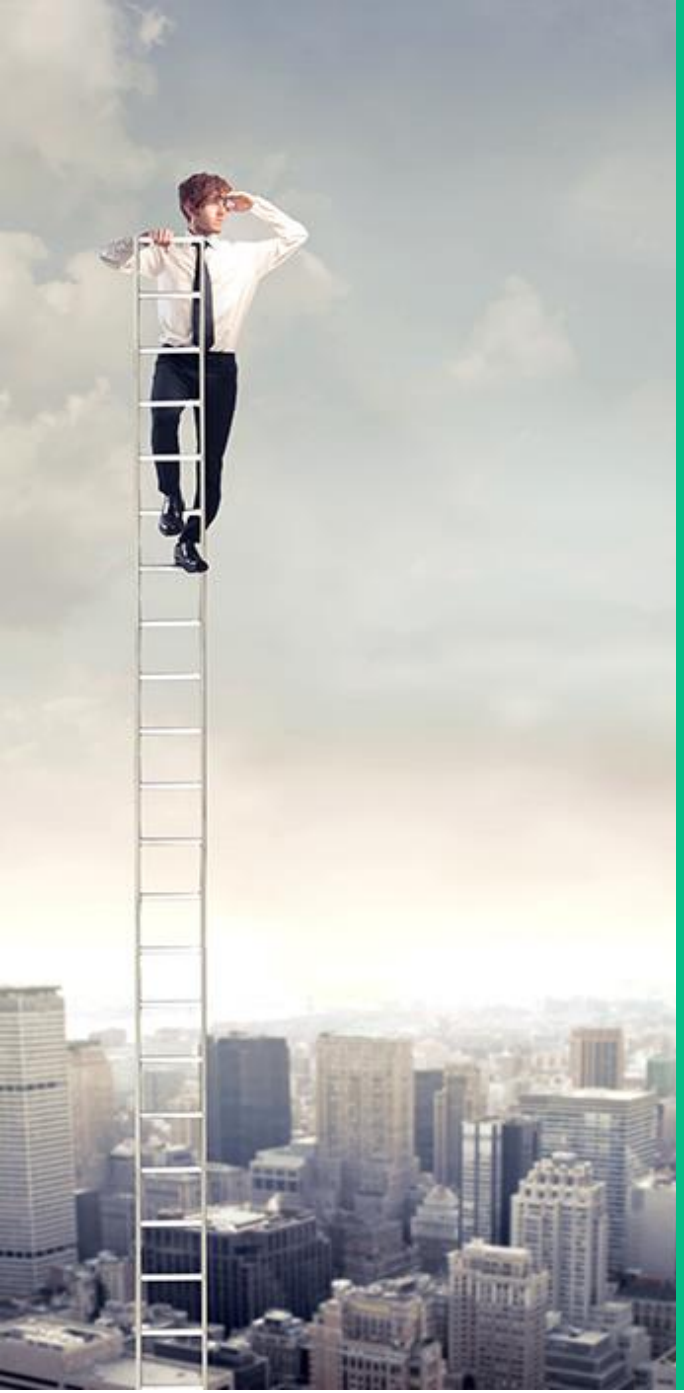


Using Cargo

Cargo



Source: https://crates.io/



Rust Crash Course

Compound Types – Tuple

The **Tuple** Type:

```
fn main() {
    let tup: (i32, f64, u8) = (500, 6.4, 1);
}
```

To get the individual values out of a tuple use pattern matching to **destructure a tuple value:**

```
fn main() {
    let tup = (500, 6.4, 1);

    let (x, y, z) = tup;

    println!("The value of y is: {y}");
}
```

Source: https://doc.rust-lang.org/book/ch03-02-data-types.html#the-tuple-type

Compound Types – Tuple

We can also access a tuple element directly by using a period (.) followed by the index of the value we want to access.

```
fn main() {
    let x: (i32, f64, u8) = (500, 6.4, 1);

    let five_hundred = x.0;

    let six_point_four = x.1;

    let one = x.2;
}
```

The tuple without any values has a special name, **unit**.

Source: https://doc.rust-lang.org/book/ch03-02-data-types.html#the-tuple-type

Compound Types – Array Type

- Every element of an array must have the **same type**.
- Arrays in Rust have a **fixed length**.

```
fn main() {
    let a = [1, 2, 3, 4, 5];
}
```

Specyfining the number of elements in the array

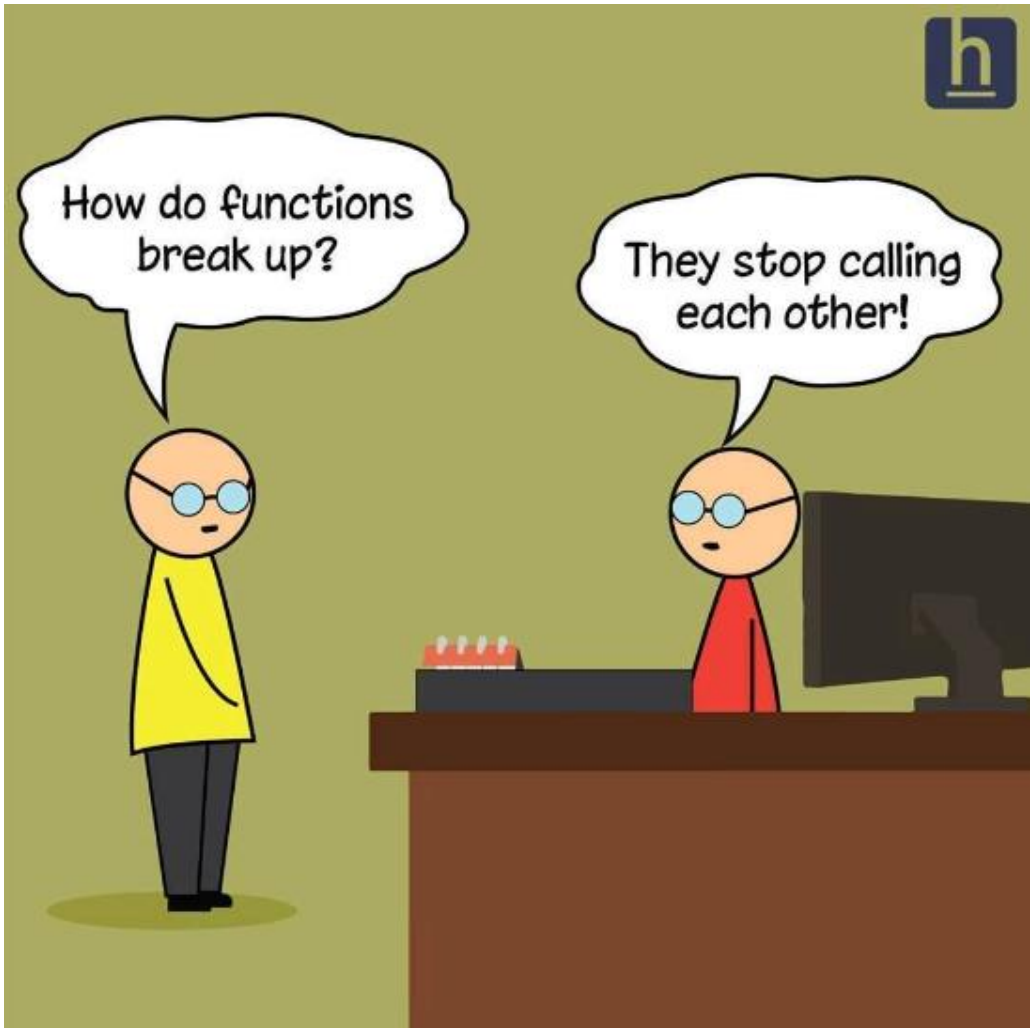
```
let a: [i32; 5] = [1, 2, 3, 4, 5];
```

Initialize an array to contain the **same value** for each element by specifying the **initial value**

```
let a = [3; 5];
```

```
let a = [3, 3, 3, 3, 3];
```

Source: https://doc.rust-lang.org/book/ch03-02-data-types.html#the-tuple-type



Source: https://pl.pinterest.com/pin/murat-pur-ontwitter--461548661804386924/

Functions

Function **definition**

```rust
fn main() {
    println!("Hello, world!");

    another_function();
}

fn another_function() {
    println!("Another function.");
}
```

Function with **parameters**

```rust
fn main() {
    another_function(5);
}

fn another_function(x: i32) {
    println!("The value of x is: {x}");
}
```

Source: https://doc.rust-lang.org/book/ch03-03-how-functions-work.html#functions

Function bodies are made up of a **series of statements** optionally ending in an expression.

- **Statements** are instructions that perform some action and do not return a value.

```rust
fn main() {
    let y = 6;
}
```

```rust
fn main() {
    let x = (let y = 6);
}
```

- **Expressions** evaluate to a resulting value.

```rust
fn main() {
    let y = {
        let x = 3;
        x + 1
    };

    println!("The value of y is: {y}");
}
```

Source: https://doc.rust-lang.org/book/ch03-03-how-functions-work.html#functions

Functions with Return Values

Declare **their type** after an arrow (**->**).

```rust
fn five() -> i32 {
    5
}

fn main() {
    let x = five();

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

```rust
fn main() {
    let x = plus_one(5);

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

fn plus_one(x: i32) -> i32 {
    x + 1
}
```

Source: https://doc.rust-lang.org/book/ch03-03-how-functions-work.html#functions

Ownership

Source: https://www.ebay.co.uk/itm/i-licked-it-so-now-its-mine-Mens-Shirt-100-cotton-Lit-funny-Ownership-claimed-/182845300076

Returning values can also transfer ownership.

```rust
fn main() {
    let s1 = gives_ownership();         // gives_ownership moves its return
                                        // value into s1

    let s2 = String::from("hello");     // s2 comes into scope

    let s3 = takes_and_gives_back(s2);  // s2 is moved into
                                        // takes_and_gives_back, which also
                                        // moves its return value into s3
} // Here, s3 goes out of scope and is dropped. s2 was moved, so nothing
  // happens. s1 goes out of scope and is dropped.

fn gives_ownership() -> String {             // gives_ownership will move its
                                             // return value into the function
                                             // that calls it

    let some_string = String::from("yours"); // some_string comes into scope

    some_string                              // some_string is returned and
                                             // moves out to the calling
                                             // function
}

// This function takes a String and returns one
fn takes_and_gives_back(a_string: String) -> String { // a_string comes into
                                                      // scope

    a_string  // a_string is returned and moves out to the calling function
}
```

Source: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html

References and Borrowing

Function that has a reference to an object as a parameter instead of taking ownership of the value

```rust
fn main() {
    let s1 = String::from("hello");

    let len = calculate_length(&s1);

    println!("The length of '{}' is {}.", s1, len);
}

fn calculate_length(s: &String) -> usize {
    s.len()
}
```

Source: https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html

Borowing

We call the action of creating a reference borrowing

```rust
fn main() {
    let s = String::from("hello");

    change(&s);
}

fn change(some_string: &String) {
    some_string.push_str(", world");
}
```

Mutable References

```rust
fn main() {
    let mut s = String::from("hello");

    change(&mut s);
}

fn change(some_string: &mut String) {
    some_string.push_str(", world");
}
```

Source: https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html

Borowing

Mutable references have one big restriction

```
let mut s = String::from("hello");

let r1 = &mut s;
let r2 = &mut s;

println!("{}, {}", r1, r2);
```

use curly brackets to create a new scope, allowing for multiple mutable references, just not *simultaneous* ones

```
let mut s = String::from("hello");

{
    let r1 = &mut s;
} // r1 goes out of scope here, so we can make a new reference with no problems.

let r2 = &mut s;
```

Source: https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html

Slices

A slice is a pointer to a block of memory.

```
let sliced_value = &data_structure[start_index..end_index]
```

Example of the usage

```
fn main(){
  let n1 = "Tutorials".to_string();

  println!("length of string is {}",n1.len());
  let c1 = &n1[4..9]; // fetches characters at 4,5,6,7, and 8 indexes
  println!("{}",c1);
  }
```

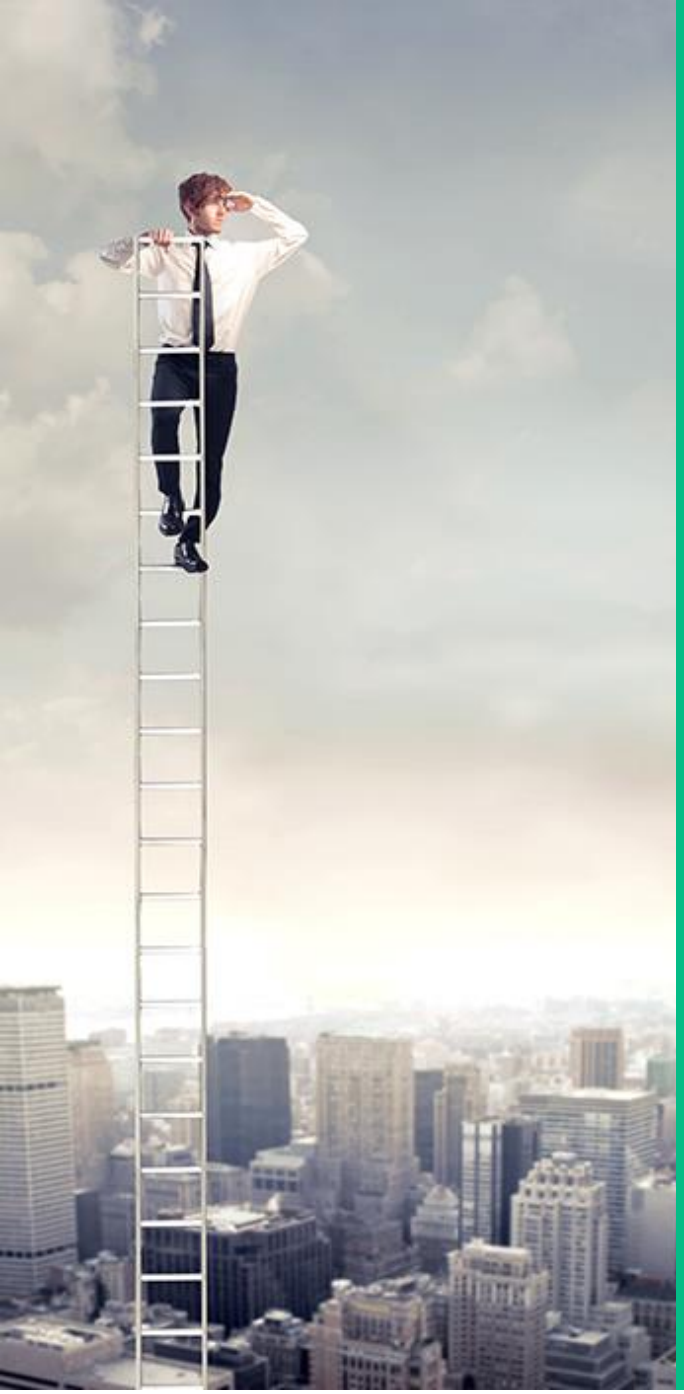
Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Slices

Mutable Slices

```
fn main(){
    let mut data = [10,20,30,40,50];
    use_slice(&mut data[1..4]);// passes references of 20, 30 and 40
    println!("{:?}",data);
}
fn use_slice(slice:&mut [i32]){
  println!("length of slice is {:?}",slice.len());
  println!("{:?}",slice);
  slice[0]=1010; // replaces 20 with 1010
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf



Syntax for working with structure

Struct

Syntaxt for declaring a struct

```
struct Name_of_structure {
    field1:data_type,
    field2:data_type,
    field3:data_type
}
```

Syntax: Initializing a structure

```
let instance_name =Name_of_structure {
    field1:value1,
    field2:value2,
    field3:value3
}; //NOTE the semicolon
```

Syntax: Accessing values in a structure

Use the dot notation to access value of a specific field.

```
instance_name.field1
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

struct

Returning struct from a function

```
fn who_is_elder (emp1:Employee,emp2:Employee)->Employee{
   if emp1.age>emp2.age {
       return emp1;
   }
   else {
     return emp2;
   }
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

struct

Method in Structure

```
struct My_struct {}
impl My_struct{     //set the method's context
    fn method_name(){  //define a method
    }
}
```

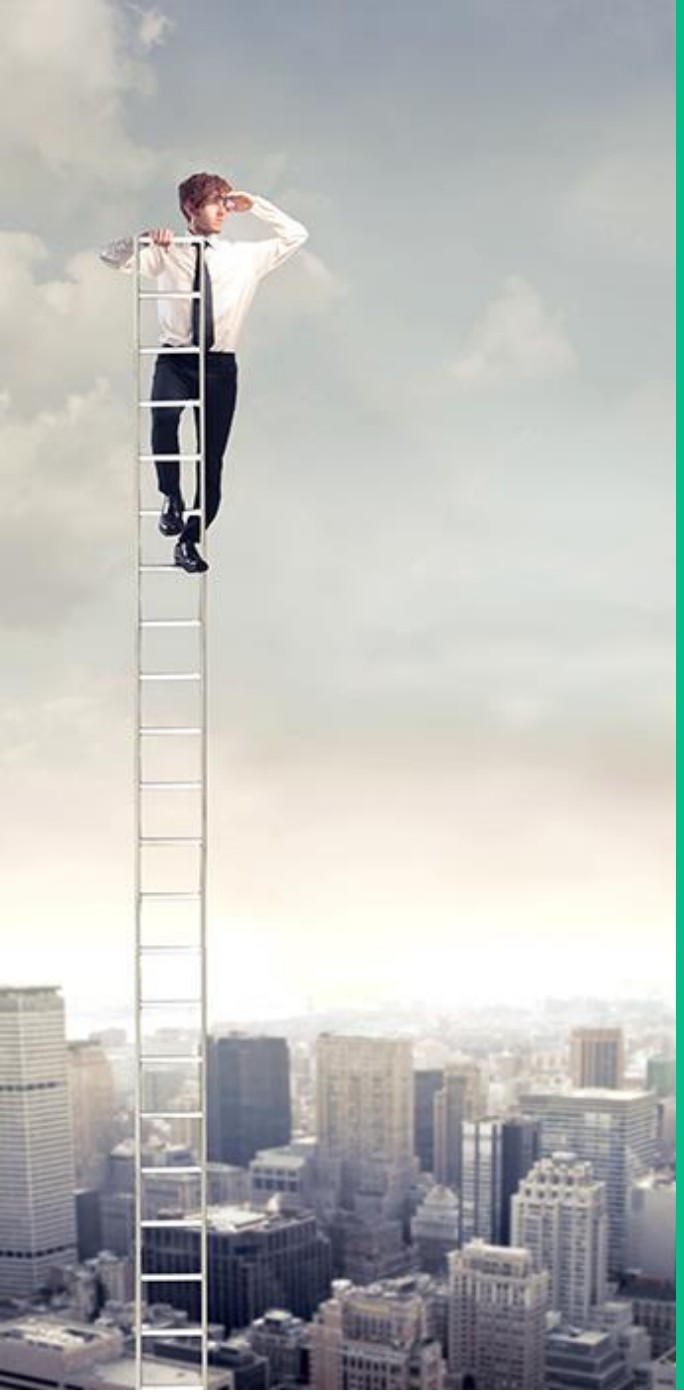
Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

struct

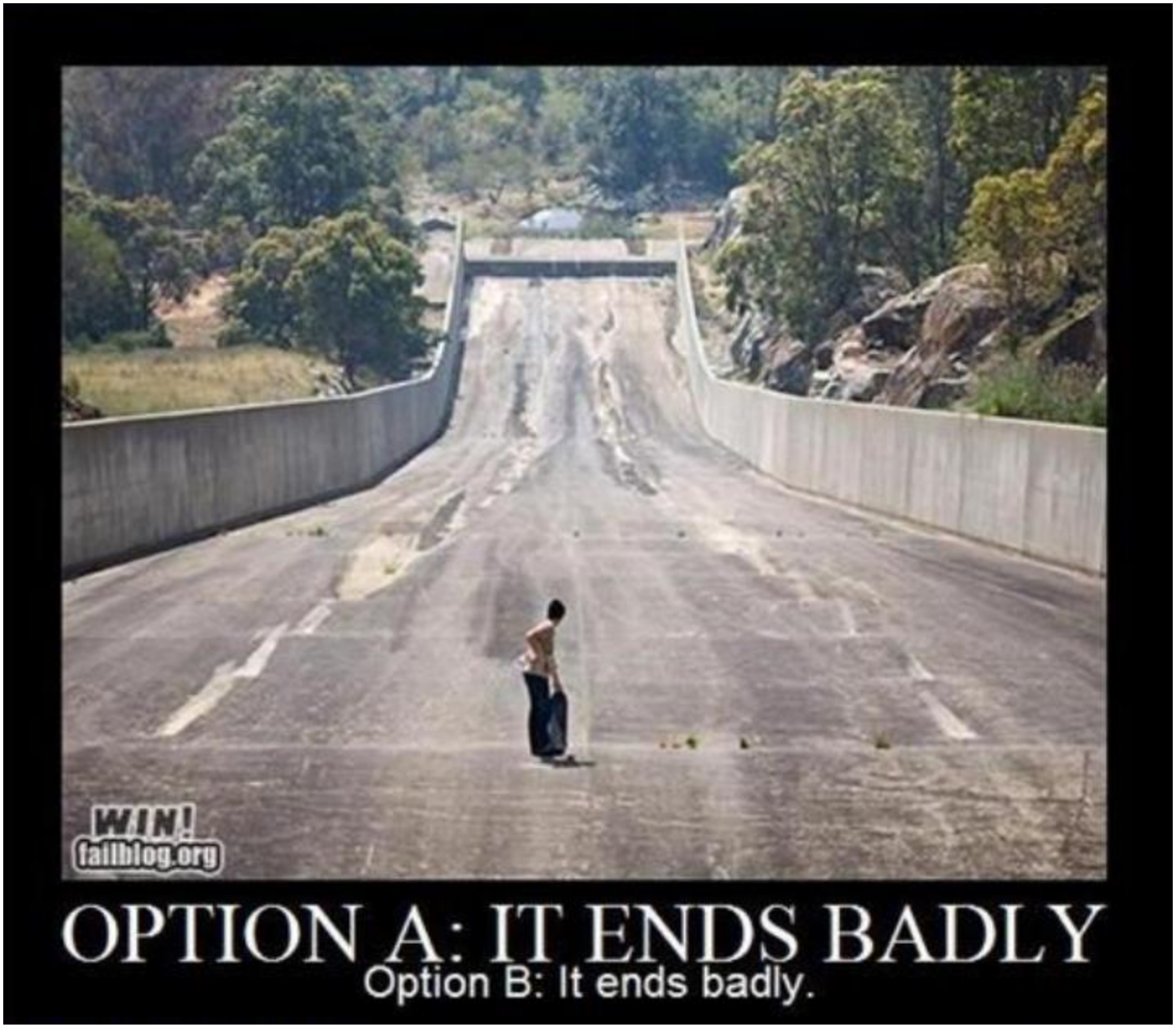
Example of method in struct

```
//define dimensions of a rectangle
struct Rectangle{
    width:u32,
    height:u32
}

//logic to calculate area of a rectangle

impl Rectangle{
    fn area(&self)->u32 {  //use the . operator to fetch the value of a field
via the self keyword
        self.width * self.height
    }
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf



Using enums



Source: https://i.chzbgr.com/full/5099889152/h068B0108/option-a

Enums

Option is a predefined enum in the Rust standard library.

```
enum Option<T> {
    Some(T),          //used to return a value
    None                 // used to return null, as Rust doesn't support the
null keyword
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Enums

Option example

```
fn main() {
    let result = is_even(3);
    println!("{:?}",result);
     println!("{:?}",is_even(30));
}

fn is_even(no:i32)->Option<bool>{
    if no%2 == 0 {
       Some(true)
       }
    else{
       None
       }
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Enums

Match Statement and Enum

```
enum CarType {
    Hatch,
    Sedan,
    SUV
}

fn print_size(car:CarType){
   match car {
       CarType::Hatch => {
           println!("Small sized car");
       },
       CarType::Sedan => {
           println!("medium sized car");
       },
       CarType::SUV =>{
           println!("Large sized Sports Utility  car");
       }
   }
}

fn main(){
    print_size(CarType::SUV);
     print_size(CarType::Hatch);
      print_size(CarType::Sedan);
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

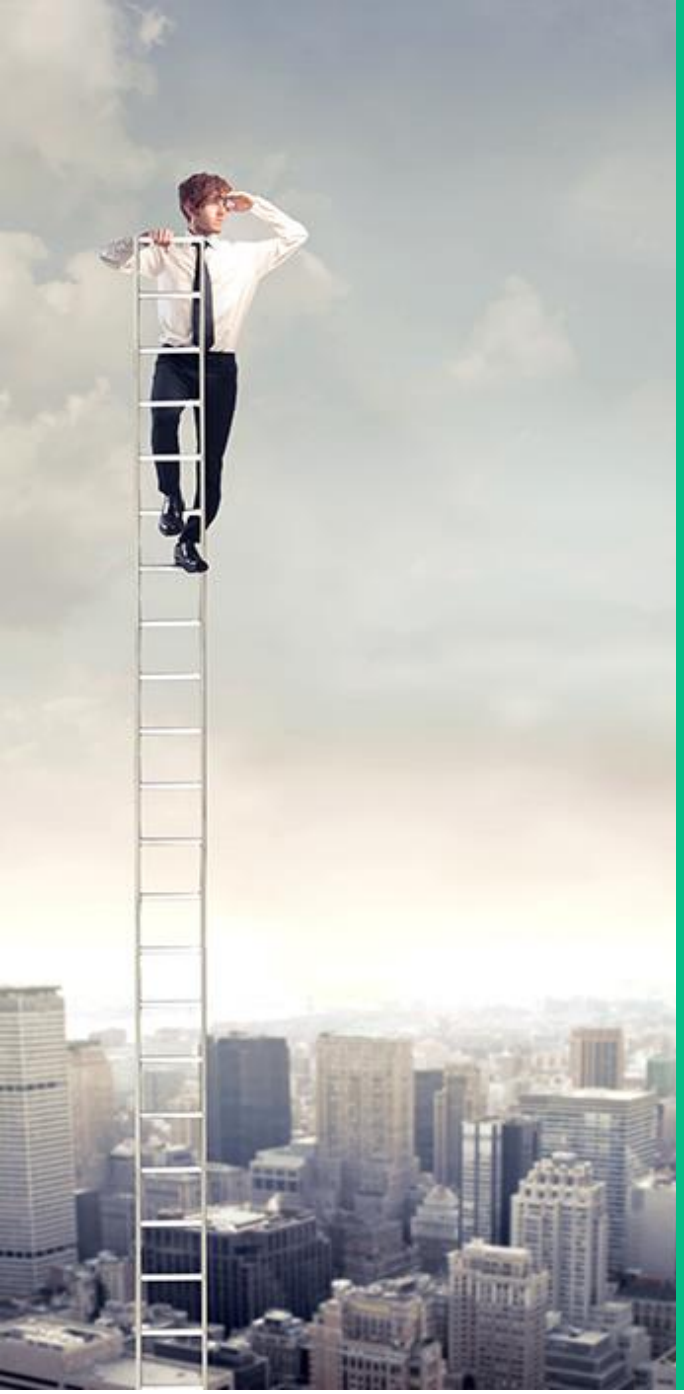
Enums

Match with Option

```
fn main() {
    match is_even(5){
        Some(data) => {
            if data==true{
                println!("Even no");
            }
        },
        None => {
            println!("not even");
        }
    }
}
```

```
fn is_even(no:i32)->Option<bool>{
    if no%2 == 0 {
        Some(true)
        }
    else{
        None
        }
 }
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf



Working with collections

Collection - Vector

Vector

- A Vector is a **resizable array**.
- It stores values in contiguous memory blocks.
- The predefined structure Vec can be used to create vectors.
- Can **grow or shrink** at runtime.
- Is a **homogeneous** collection.
- Stores data as sequence of elements in a **particular** order.
- Will only **append** values to the end.
- Memory for a Vector is allocated in the **heap**.

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Collection - Vector

Creating a Vector

```
let mut instance_name = Vec::new();
let  vector_name = vec![val1,val2,val3]
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Collection - Vector

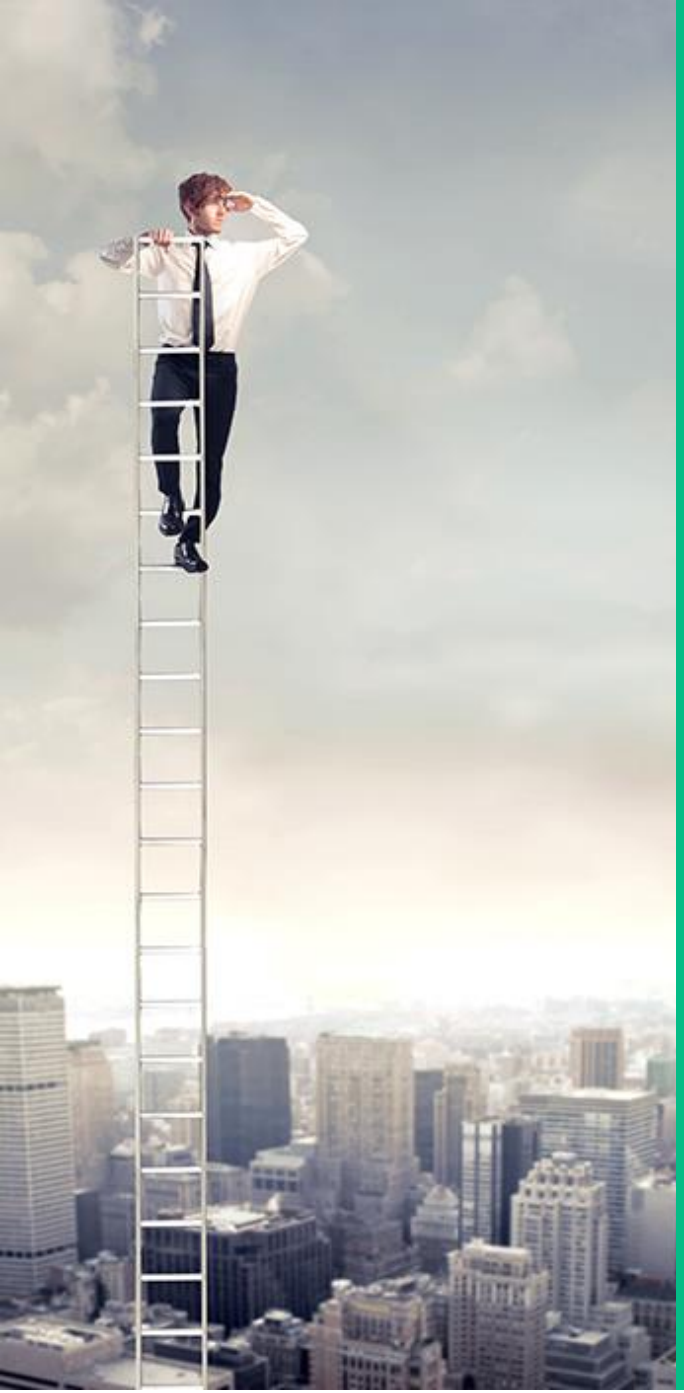
Creating a Vector - new()

```
fn main() {
    let mut v = Vec::new();
    v.push(20);
    v.push(30);
    v.push(40);

    println!("size of vector is :{}",v.len());
    println!("{:?}",v);
}
```

Creating a Vector - vec! Macro

```
fn main() {
    let  v = vec![1,2,3];
    println!("{:?}",v);
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf



Handing Error

Source: https://twitter.com/tgoecke/status/580472825439522816

Handling Errors

Result Enum and Recoverable Errors

```
enum Result<T,E> {
    OK(T),
    Err(E)
}
```

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Handling Errors

```
fn main(){

    let result = is_even(13);

    match result {

        Ok(d)=>{
            println!("no is even {}",d);
        },
        Err(msg)=>{
            println!("Error msg is {}",msg);
        }

    }
      println!("end of main");
}
```

The is_even function returns an error if the number is not an even number. The main() function handles this error.

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

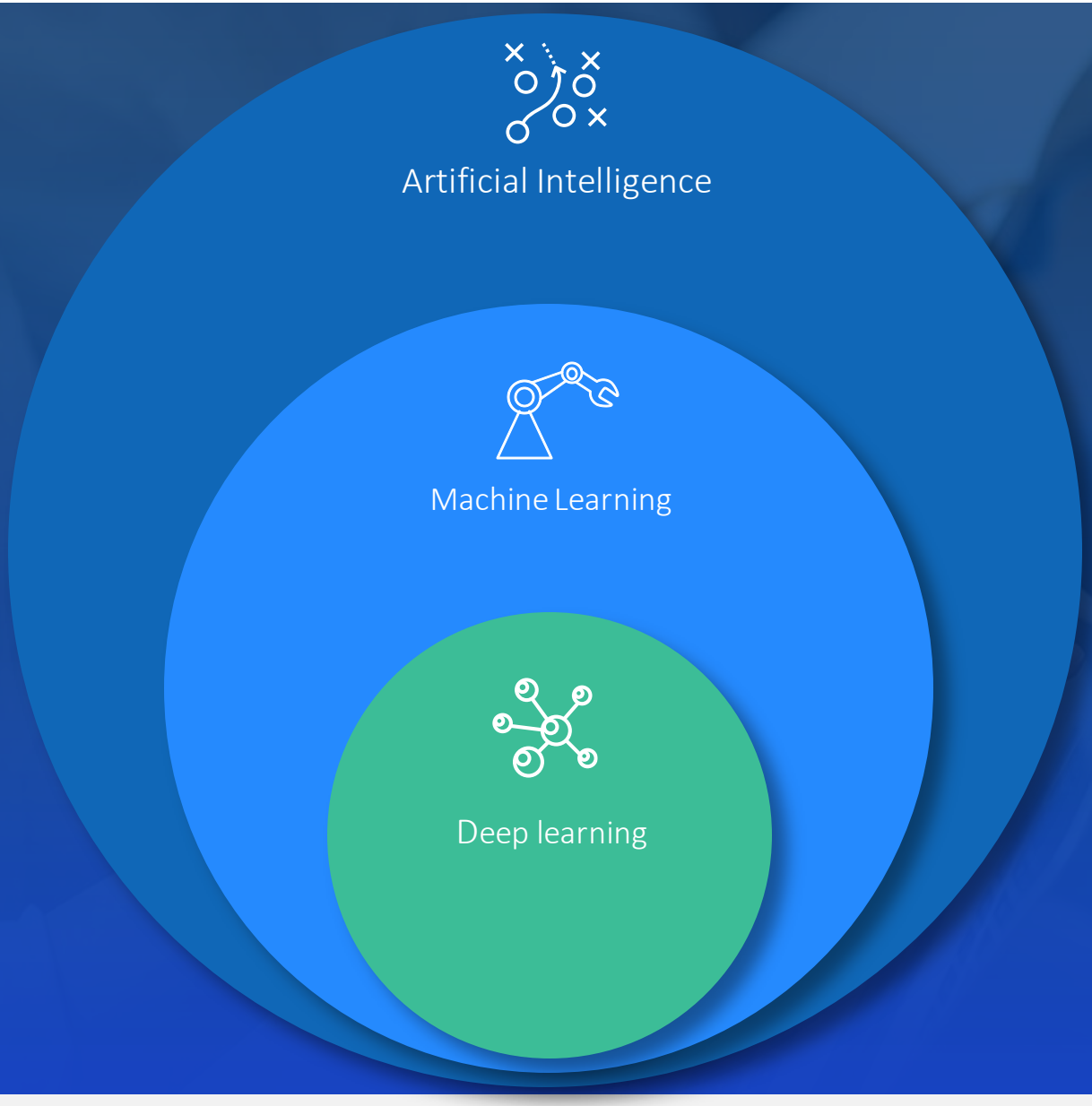
Handling Errors

```
fn is_even(no:i32)->Result<bool,String>{

   if no%2==0 {
       return Ok(true);
    }
    else  {
     return Err("NOT_AN_EVEN".to_string());
    }

 }
```

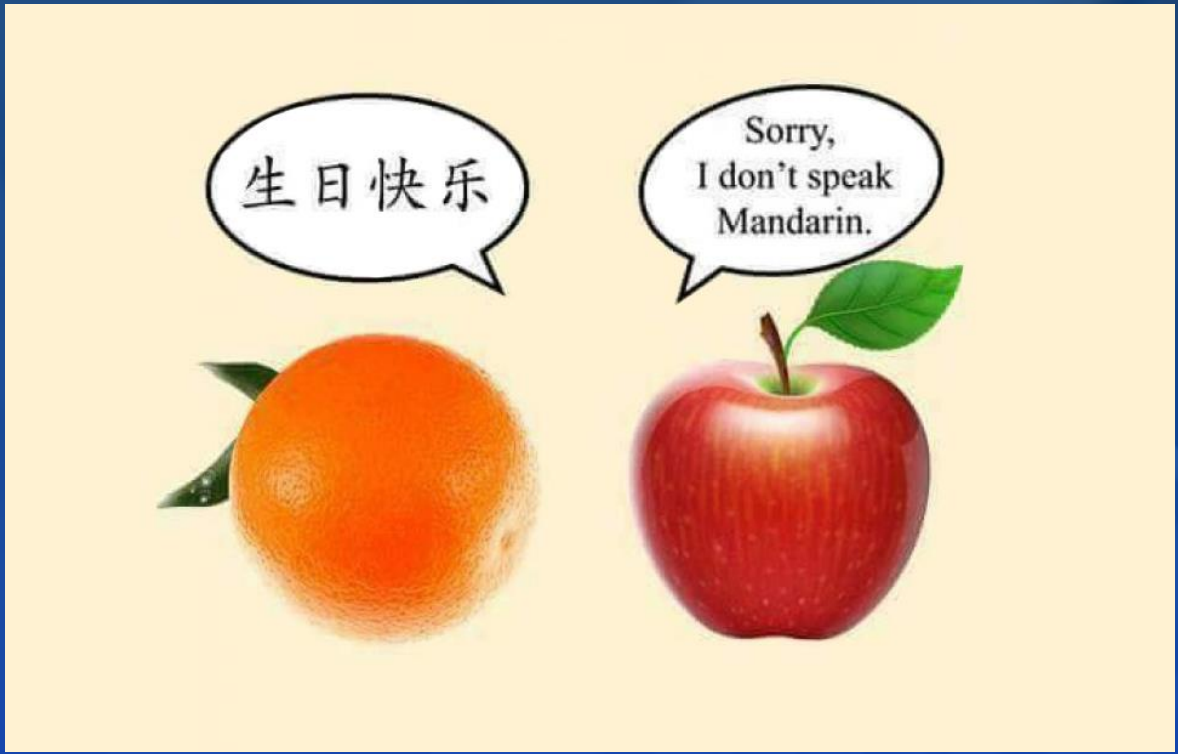
The is_even function returns an error if the number is not an even number. The main() function handles this error.

Source: https://www.tutorialspoint.com/rust/rust_tutorial.pdf

Artificial Intelligence

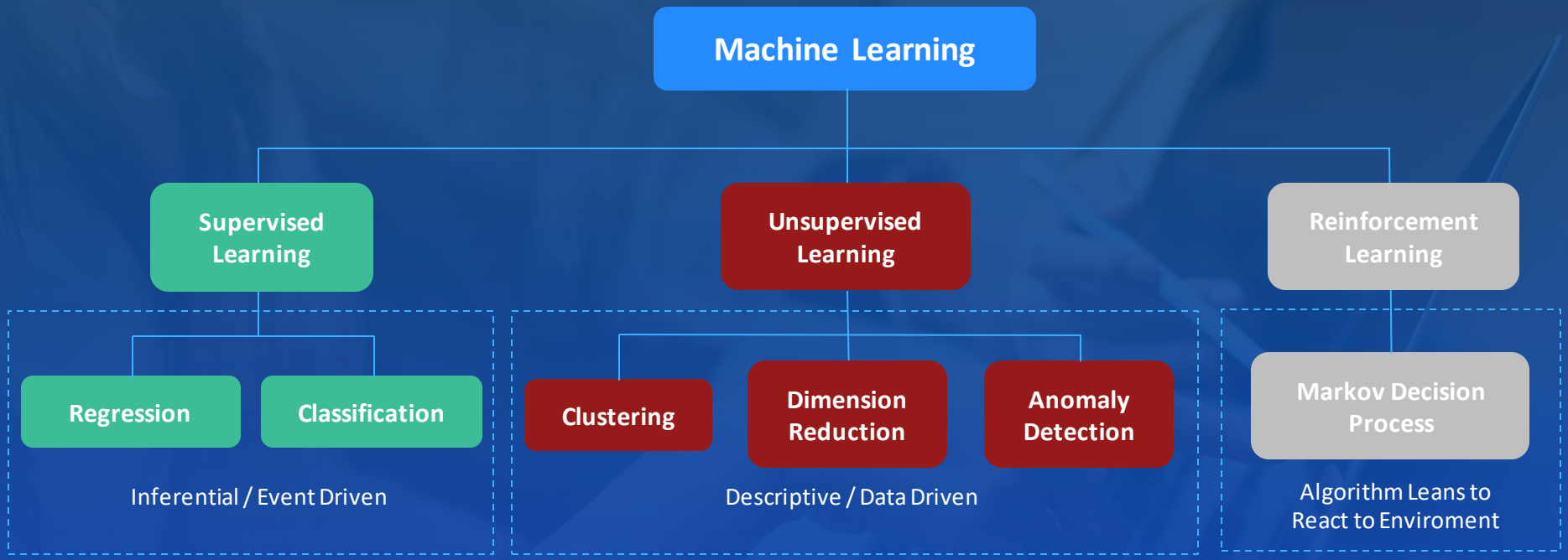


Learning



Source: http://www.obrazki.jeja.pl/359459,mandarynski-chinski.html

Machine Learning



Source: Mastering Machine Learning with Python in Six Steps

Machine Learning

Supervised Learning

Reinforcemnet Learning

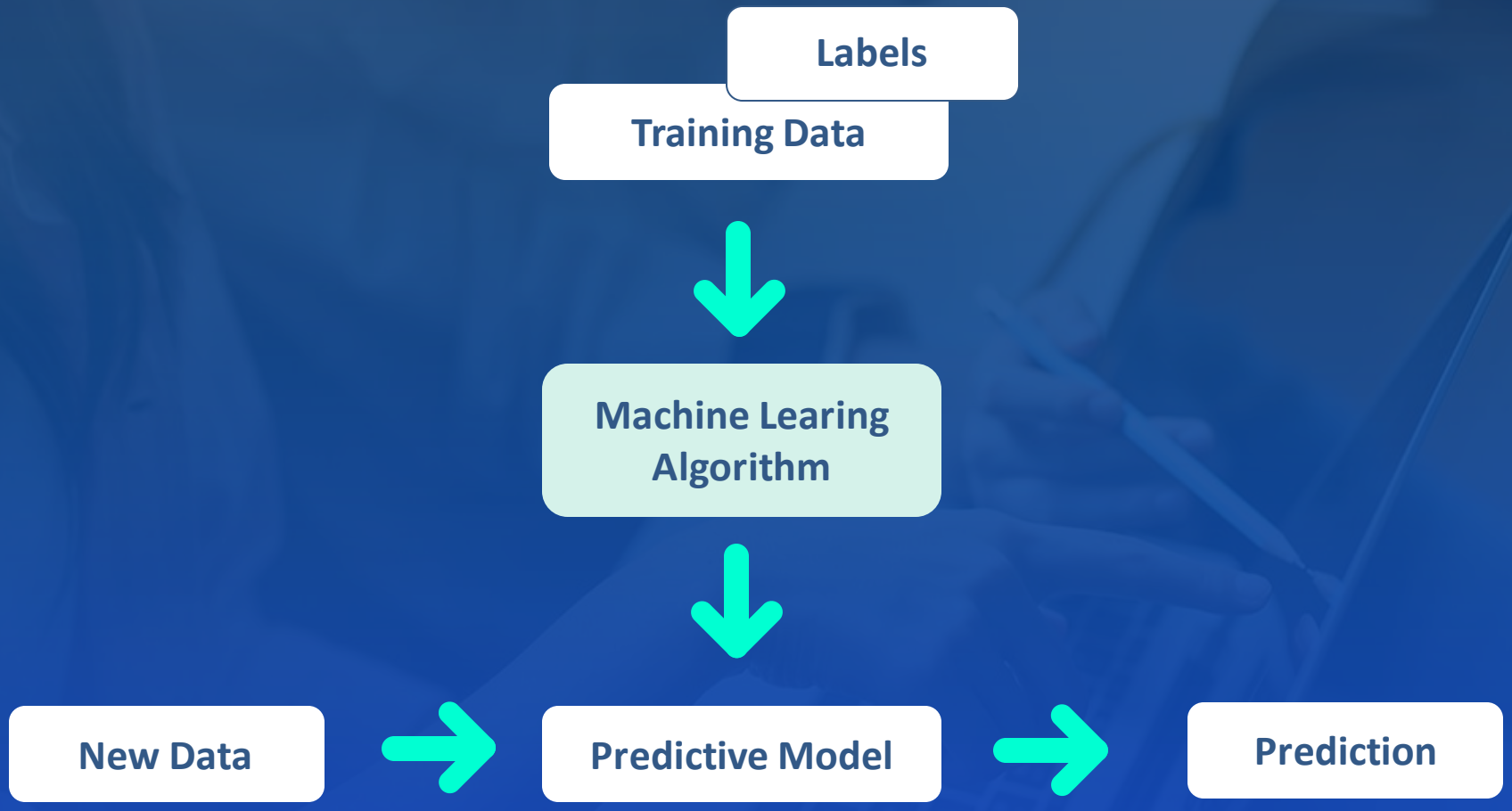
Unsupervised Learning

Supervised Learning

Unsupervised Learning

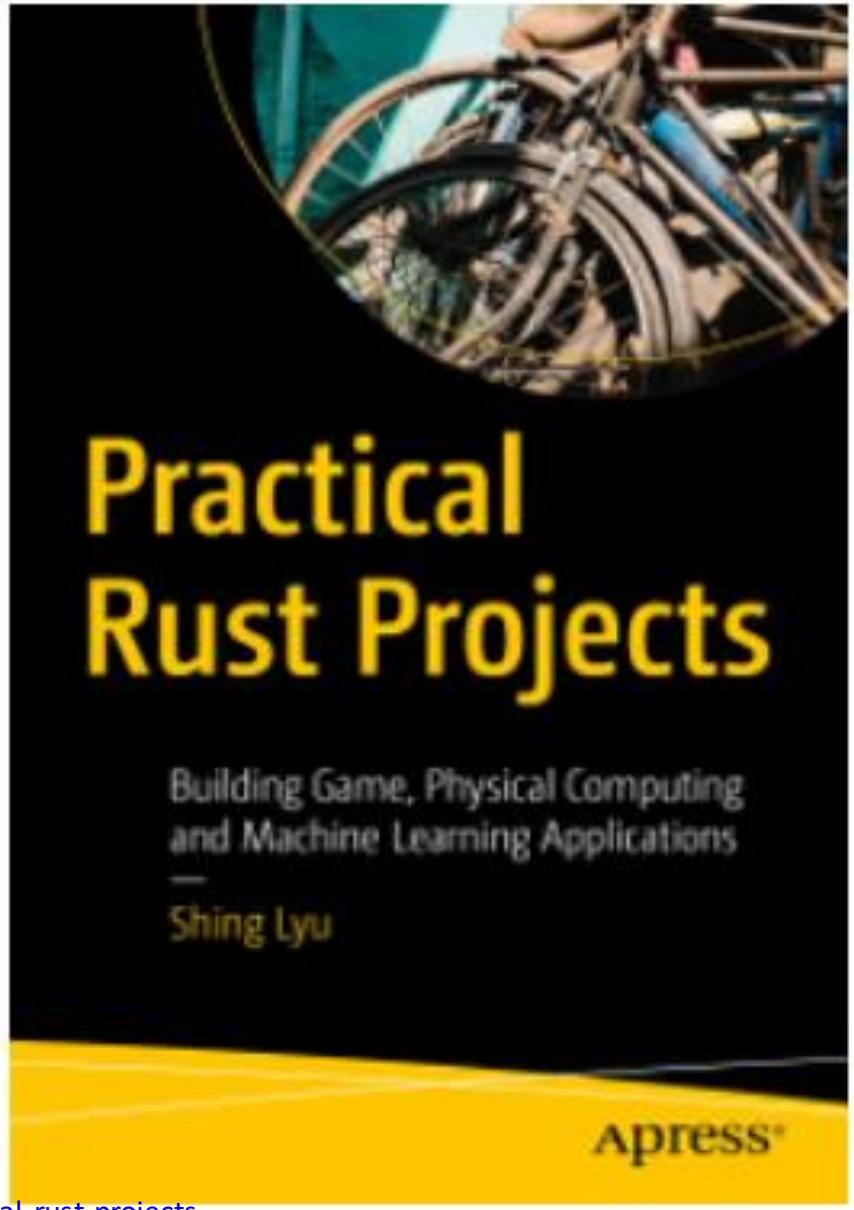
Source: Machine Learning for Audio, Image and Video Analysis

Machine Learning



Source: Python Machine Learning, Sebastian Raschka, 2015 Packt Publishing

Next Steps



Source: https://github.com/Apress/practical-rust-projects

AI with Rust - Cluster

```
extern crate rusty_machine;
extern crate rand;

use std::io;
use std::error::Error;

use rusty_machine::linalg::{Matrix, BaseMatrix};
use rusty_machine::learning::k_means::KMeansClassifier;
use rusty_machine::learning::UnSupModel;

const CLUSTER_COUNT: usize = 3;

fn read_data_from_stdin() -> Result<Matrix<f64>, Box<dyn Error>> {
    let mut reader = csv::Reader::from_reader(io::stdin());
    let mut data: Vec<f64>= vec![];
    for result in reader.records() {
        let record = result?;
        data.push(record[0].parse().unwrap());
        data.push(record[1].parse().unwrap());
    }

    Ok(Matrix::new(&data.len() / 2, 2, data))
}
// (…)
```

Source: https://github.com/Apress/practical-rust-projects

AI with Rust – Cluster

```
// (…)

fn export_result_to_stdout(samples: Matrix<f64>, classes: Vec<usize>) -> Result<(), Box<dyn Error>> {
    let mut writer = csv::Writer::from_writer(io::stdout());
    writer.write_record(&["height", "length", "class"])?;
    for sample in samples.iter_rows().zip(classes) {
        writer.serialize(sample)?;
    }
    Ok(())
}

fn main() {

    let samples = read_data_from_stdin().unwrap();

    let mut model = KMeansClassifier::new(CLUSTER_COUNT);
    model.train(&samples).unwrap();

    let classes = model.predict(&samples).unwrap();

    export_result_to_stdout(samples, classes.into_vec()).unwrap();
}
```

Source: https://github.com/Apress/practical-rust-projects

Plot Data

```rust
use std::error::Error;
use std::io;
use gnuplot::{Figure, Caption, Graph};
use gnuplot::AxesCommon;

fn main() -> Result<(), Box<dyn Error>>{
    let mut x: Vec<f64> = Vec::new();
    let mut y: Vec<f64> = Vec::new();

    let mut reader = csv::Reader::from_reader(io::stdin());
    for result in reader.records() {
        let record = result?;
        x.push(record[0].parse().unwrap());
        y.push(record[1].parse().unwrap());
    }

    let mut fg = Figure::new();
    fg.axes2d()
        .set_title("Cat body measurements", &[])
        .set_legend(Graph(0.9), Graph(0.1), &[], &[])
        .set_x_label("height (cm)", &[])
        .set_y_label("length (cm)", &[])
        .points(x, y, &[Caption("Cat")]);
    fg.show();
    Ok(())
}
```

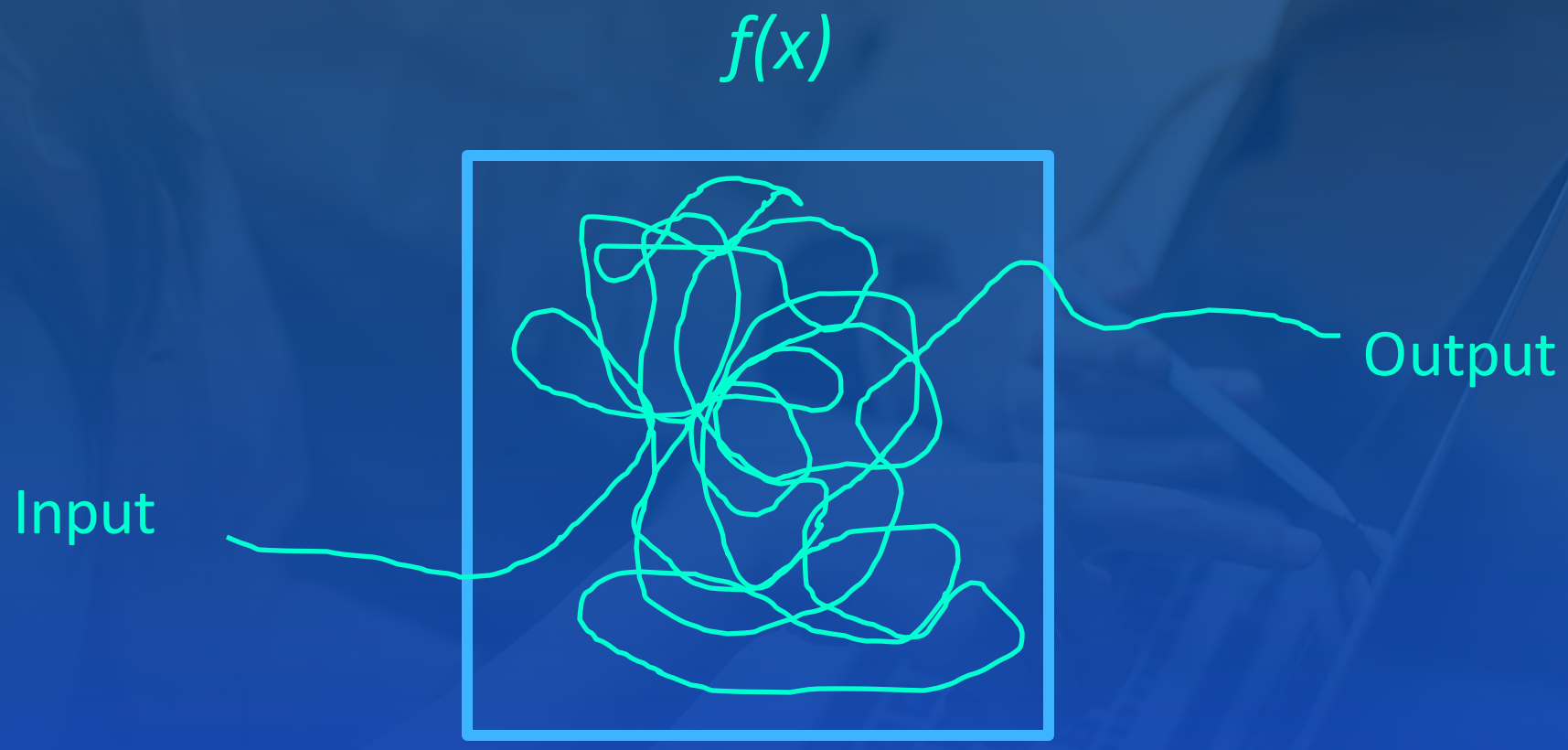
Source: https://github.com/Apress/practical-rust-projects

Plot Data

```rust
use std::error::Error;
use std::io;
use gnuplot::{Figure, Caption, Graph, Color, PointSymbol};
use gnuplot::AxesCommon;

fn main() -> Result<(), Box<dyn Error>>{
    let mut x: [Vec<f64>; 3] = [Vec::new(), Vec::new(), Vec::new()];
    let mut y: [Vec<f64>; 3] = [Vec::new(), Vec::new(), Vec::new()];

    let mut reader = csv::Reader::from_reader(io::stdin());
    for result in reader.records() {
        let record = result?;
        let class:usize = record[2].parse().unwrap();
        x[class].push(record[0].parse().unwrap());
        y[class].push(record[1].parse().unwrap());
    }

    let mut fg = Figure::new();
    fg.axes2d()
            .set_title("Cat breed classification result", &[])
            .set_legend(Graph(0.9), Graph(0.1), &[], &[])
            .set_x_label("height (cm)", &[])
            .set_y_label("length (cm)", &[])
            .points(
                    &x[0],
                    &y[0],
                    &[Caption("Cat breed 1"), Color("red"), PointSymbol('+')],
            )
            .points(
                    &x[1],
                    &y[1],
                    &[Caption("Cat breed 2"), Color("green"), PointSymbol('x')],
            )
            .points(
                    &x[2],
                    &y[2],
                    &[Caption("Cat breed 3"), Color("blue"), PointSymbol('o')],
            );
    fg.show();
    Ok(())
}
```

Source: https://github.com/Apress/practical-rust-projects

Neural Network



Source: Thoughtful Machine Learnig in Python, Matthew Kirk

Neural Network

```rust
extern crate rusty_machine;
extern crate rand;

use rusty_machine::linalg::{Matrix, BaseMatrix};

use rand::thread_rng;
use rand::distributions::Distribution; // for using .sample()
use rand_distr::Normal; // splitted from rand since 0.7
use std::io;
use serde::Serialize;

// settings
const CENTROIDS:[f64;4] =  [ // Height, length
    61.0, 99.5, // German Shepherd dog
    22.5, 40.5, // Persian cat
];

const NOISE:f64 = 1.8;
const SAMPLES_PER_CENTROID: usize = 2000;

#[derive(Debug, Serialize)]
struct Sample {
    height: f64,
    length: f64,
    category_id: usize
}
```

Source: https://github.com/Apress/practical-rust-projects/blob/master/machine-learning/cat-or-dog-nnet/src4/bin/generate_data.rs

Neural Network

```rust
fn generate_data(centroids: &Matrix<f64>,
                 points_per_centroid: usize,
                 noise: f64)
                 -> Vec<Sample> {
    assert!(centroids.cols() > 0, "Centroids cannot be empty.");
    assert!(centroids.rows() > 0, "Centroids cannot be empty.");
    assert!(noise >= 0f64, "Noise must be non-negative.");
    let mut samples = Vec::with_capacity(points_per_centroid);

    let mut rng = thread_rng();
    let normal_rv = Normal::new(0f64, noise).unwrap();

    for _ in 0..points_per_centroid {
        // Generate points from each centroid
        for (centroid_id, centroid) in centroids.iter_rows().enumerate() {
            let mut point = Vec::with_capacity(centroids.cols());
            for feature in centroid.iter() {
                point.push(feature + normal_rv.sample(&mut rng));
            }

            samples.push(Sample {
                height: point[0],
                length: point[1],
                category_id: centroid_id,
            });
        }
    }

    samples
}
```

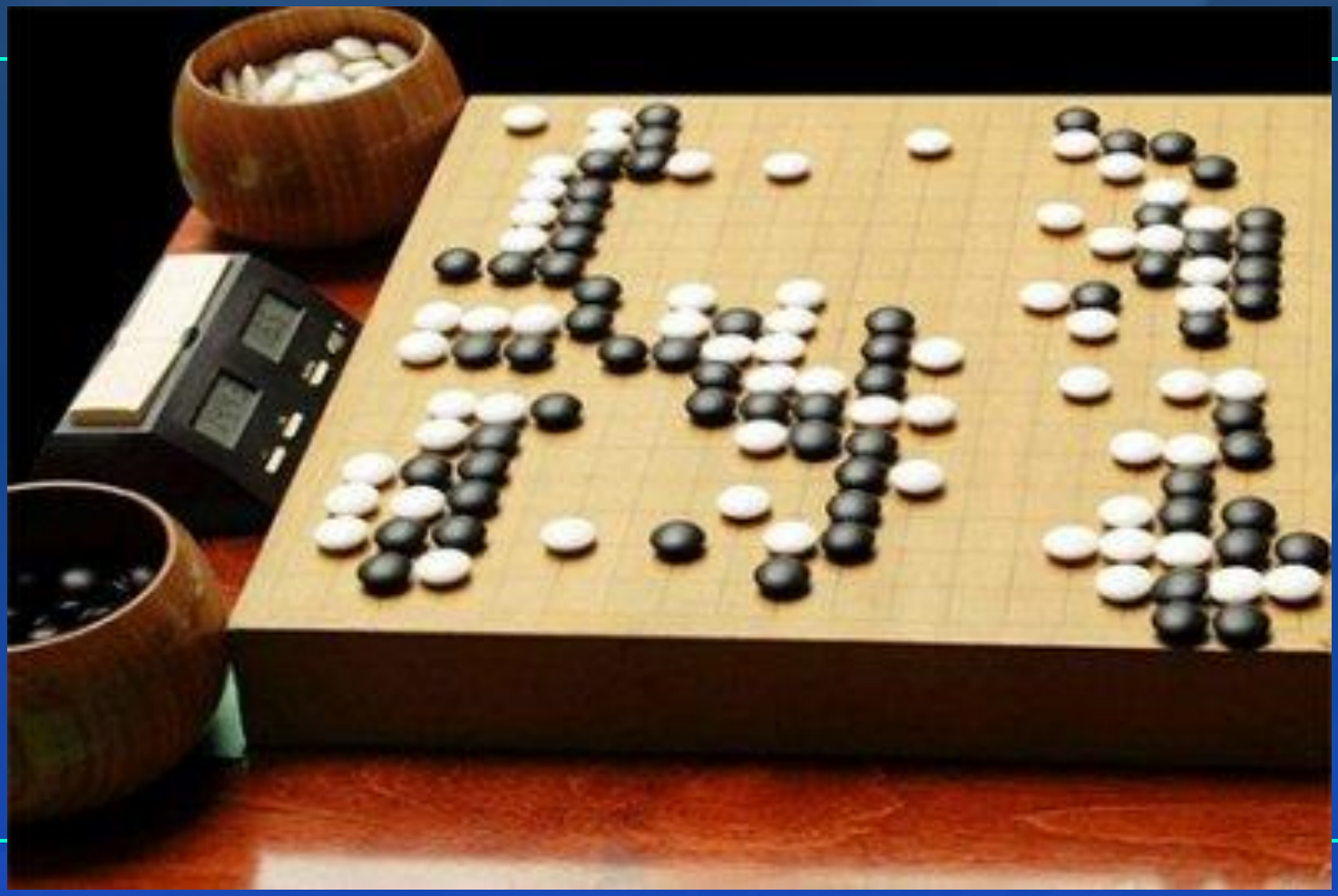
Source: https://github.com/Apress/practical-rust-projects/blob/master/machine-learning/cat-or-dog-nnet/src4/bin/generate_data.rs

Neural Network

```rust
fn main() -> Result<(), std::io::Error> {
    let centroids = Matrix::new(2, 2, CENTROIDS.to_vec());

    let samples = generate_data(&centroids, SAMPLES_PER_CENTROID, NOISE);

    let mut writer = csv::Writer::from_writer(io::stdout());
    // serialize will generate the column header automatically
    for sample in samples.iter() {
        writer.serialize(sample)?;
    }
    Ok(())
}
```

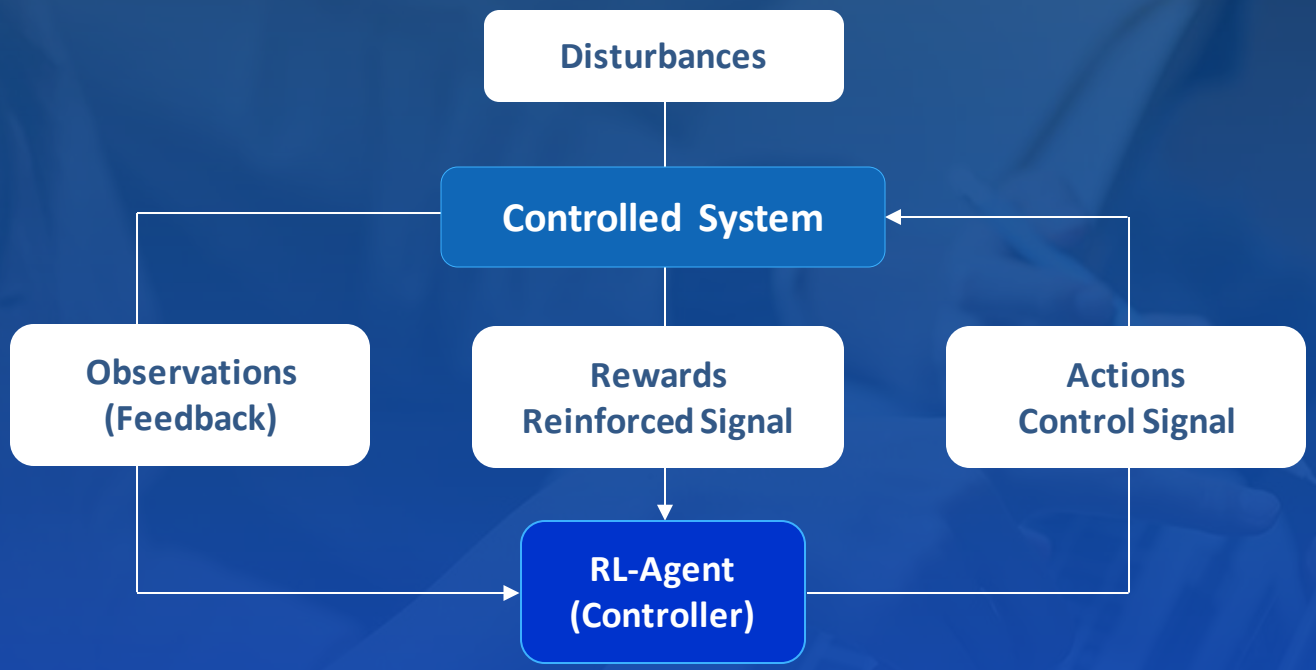
Source: https://github.com/Apress/practical-rust-projects/blob/master/machine-learning/cat-or-dog-nnet/src4/bin/generate_data.rs

Machine Learning



Source: https://www.usgo.org/what-go

High-level flow of Reinforcement Learning

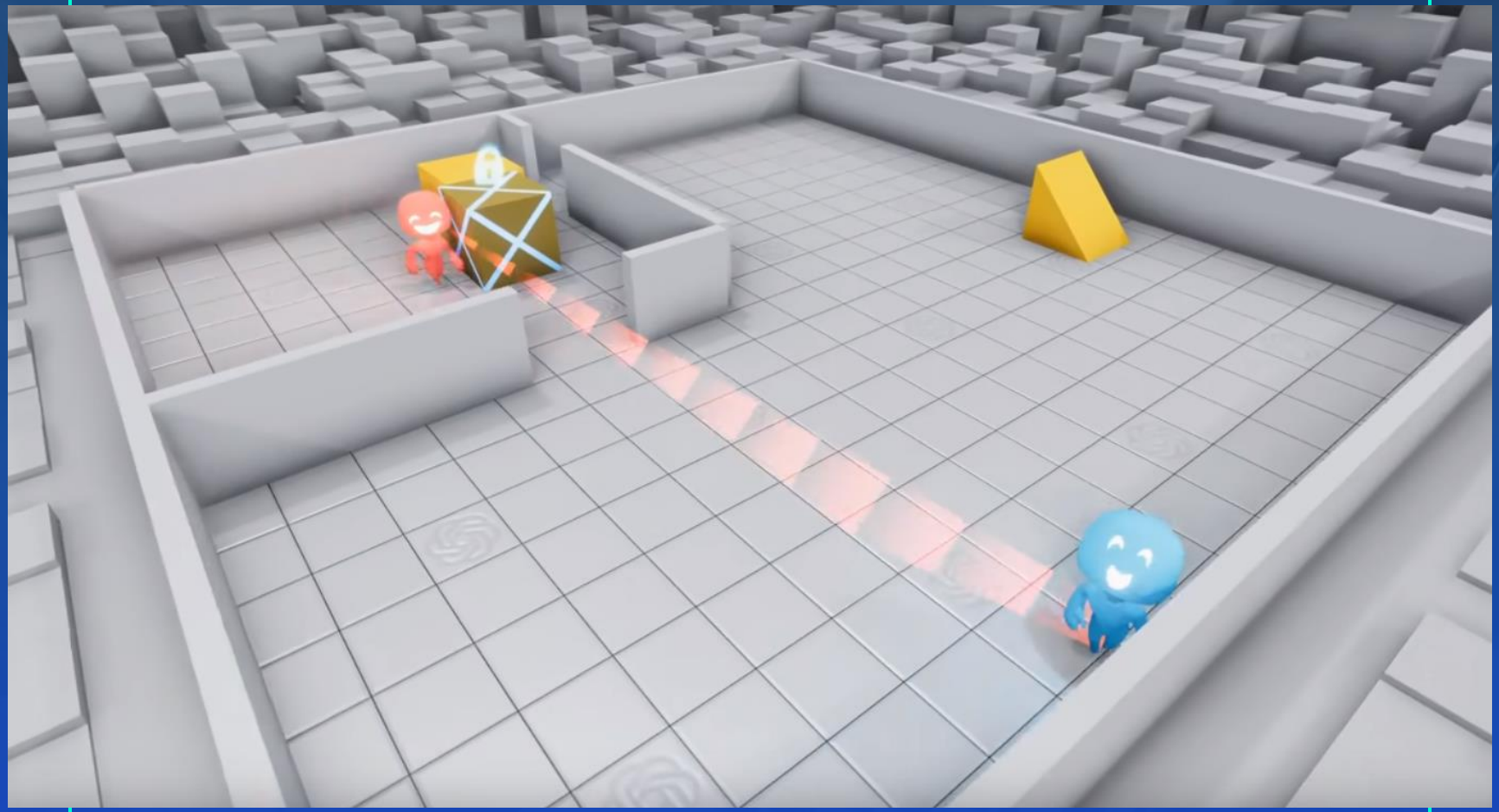


Source: Efficient Learning Machines

Machine Learning

Source: https://www.youtube.com/watch?v=UHU8ICctC-Q

Machine Learning



Source: https://openai.com/blog/emergent-tool-use/

Reinforcment Learning in Rust

```
let env = MountainCar::default();
let n_actions = env.action_space().card().into();

let mut rng = StdRng::seed_from_u64(0);
let (mut ql, policy) = {
    let basis = Fourier::from_space(5, env.state_space()).with_bias();
    let q_func = make_shared(LFA::vector(basis, SGD(0.001), n_actions));
    let policy = Greedy::new(q_func.clone());

    (QLearning {
        q_func,
        gamma: 0.9,
    }, policy)
};

for e in 0..200 {
    // Episode loop:
    let mut j = 0;
    let mut env = MountainCar::default();
    let mut action = policy.sample(&mut rng, env.emit().state());

    for i in 0.. {
        // Trajectory loop:
        j = i;

        let t = env.transition(action);

        ql.handle(&t).ok();
        action = policy.sample(&mut rng, t.to.state());

        if t.terminated() {
            break;
        }
    }

    println!("Batch {}: {} steps...", e + 1, j + 1);
}

let traj = MountainCar::default().rollout(|s| policy.mode(s), Some(500));

println!("OOS: {} states...", traj.n_states());
```

Source: https://github.com/tspooner/rsrl

Thank you
for your attention!