

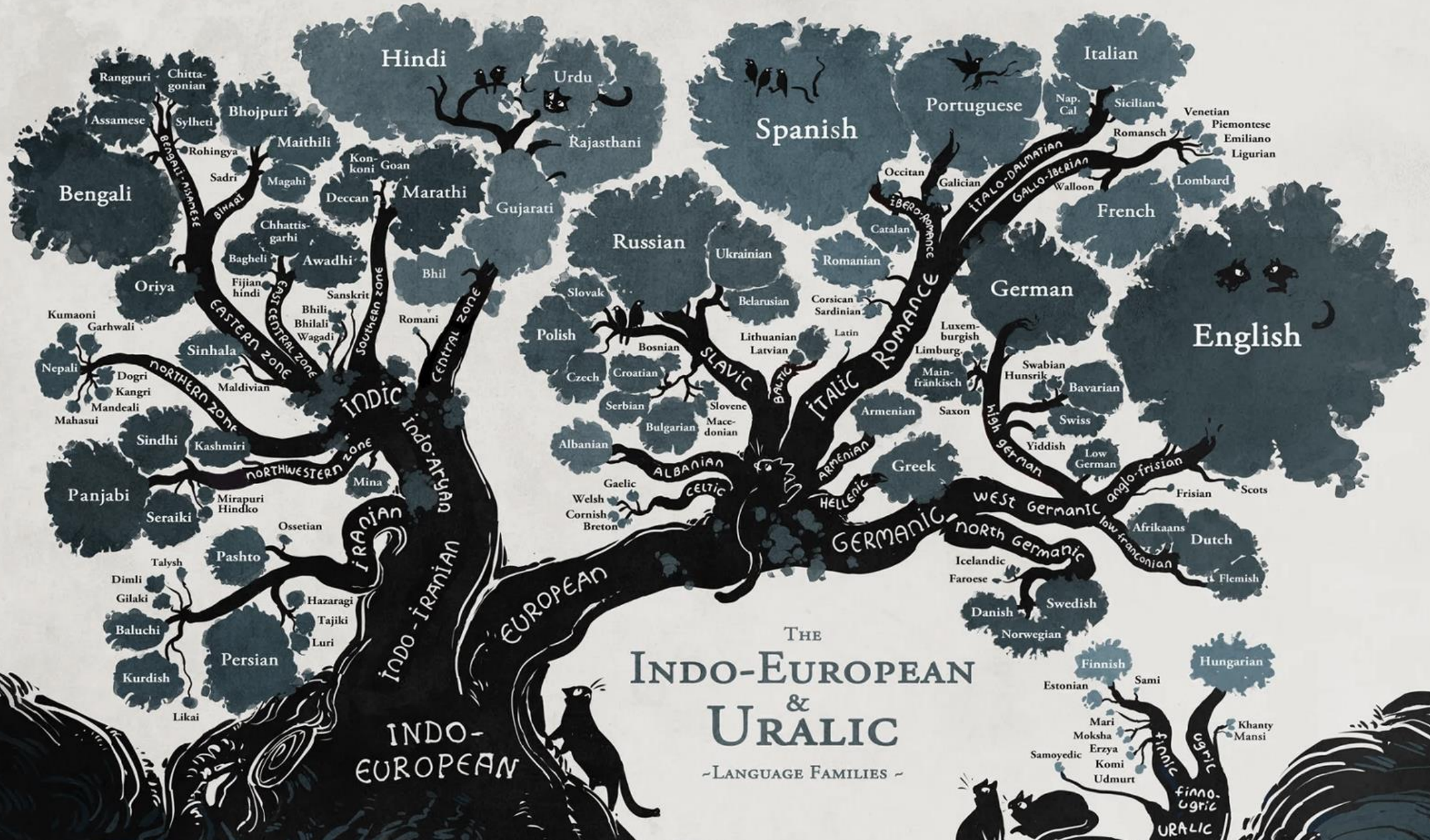
Offensive Development in Modern Languages

Cas van Cooten

OrangeCon 2024

SLIDES AVAILABLE






Graphic: Minna Sundberg

01 | About

```
[cas@OrangeCon ~]$ whoami
```


- Offensive Security enthusiast, Red Team operator & self-proclaimed “Malware Linguist”
- Likes building offensive tooling in modern languages ( Rust, Nim, Go & Python)
- Publishes OST and various offsec-related repositories on Github (for example **Nimplant**)
- Semi-pro shitposter on Twitter



Cas van Cooten

 casvancooten.com

 [@chvancooten](https://twitter.com/chvancooten)

 [chvancooten](https://github.com/chvancooten)

 [/in/chvancooten](https://www.linkedin.com/in/chvancooten)



02 | Offensive Development

To develop or not to develop, that is the question

Developing in-house is ultimately a business decision



Purchase

Purchase operations-ready
commercial tools



Adapt

Modify open-source
projects to fit your needs



Develop

Build your own tools from
scratch

02 | Offensive Development

“Offensive Development” vs. “Malware Development”



...and much more

02 | Offensive Development

Putting the 'Development' in Offensive Development

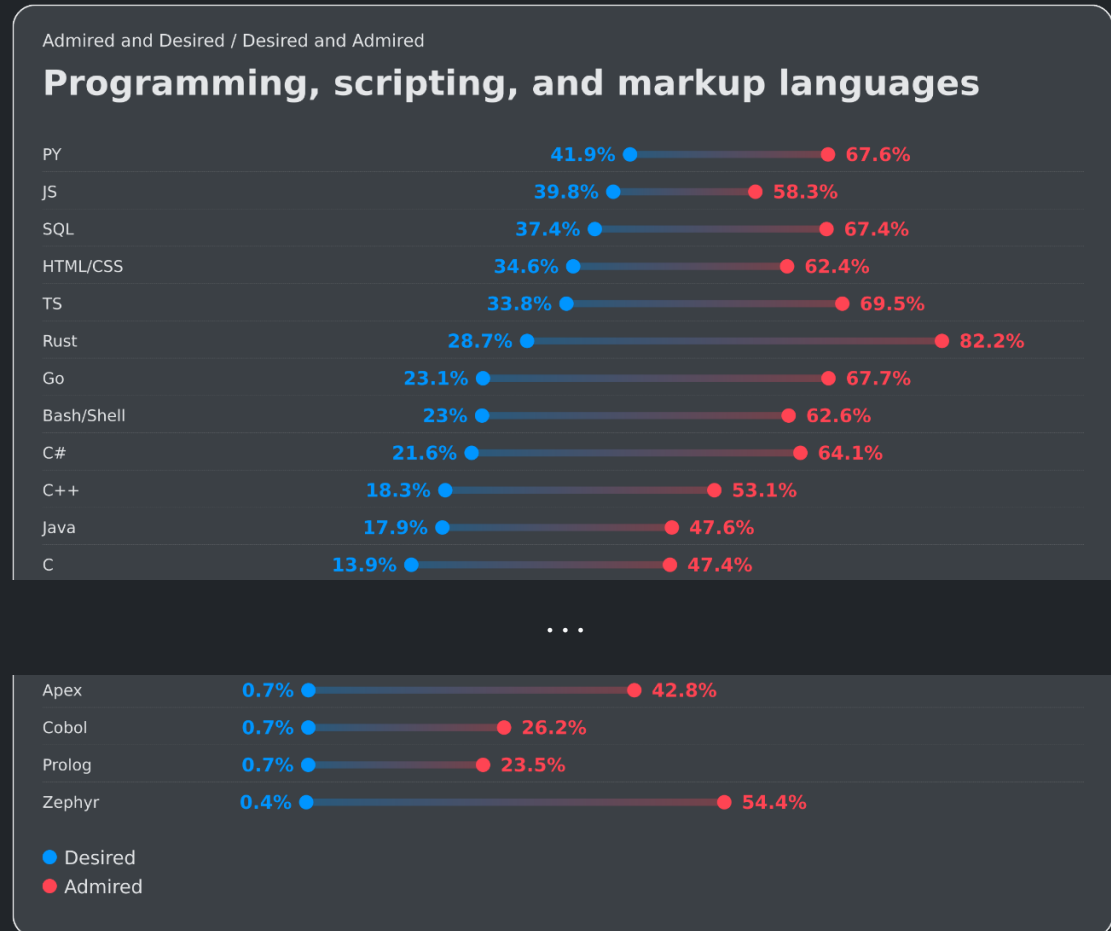
- “Working code is better than perfect code”, **BUT**
- Bad code has a real impact:
 - Readability & Maintainability
 - Stability
 - Security
- Adopting a **‘developer mindset’** is easier than you think
- AI can be of great help here!

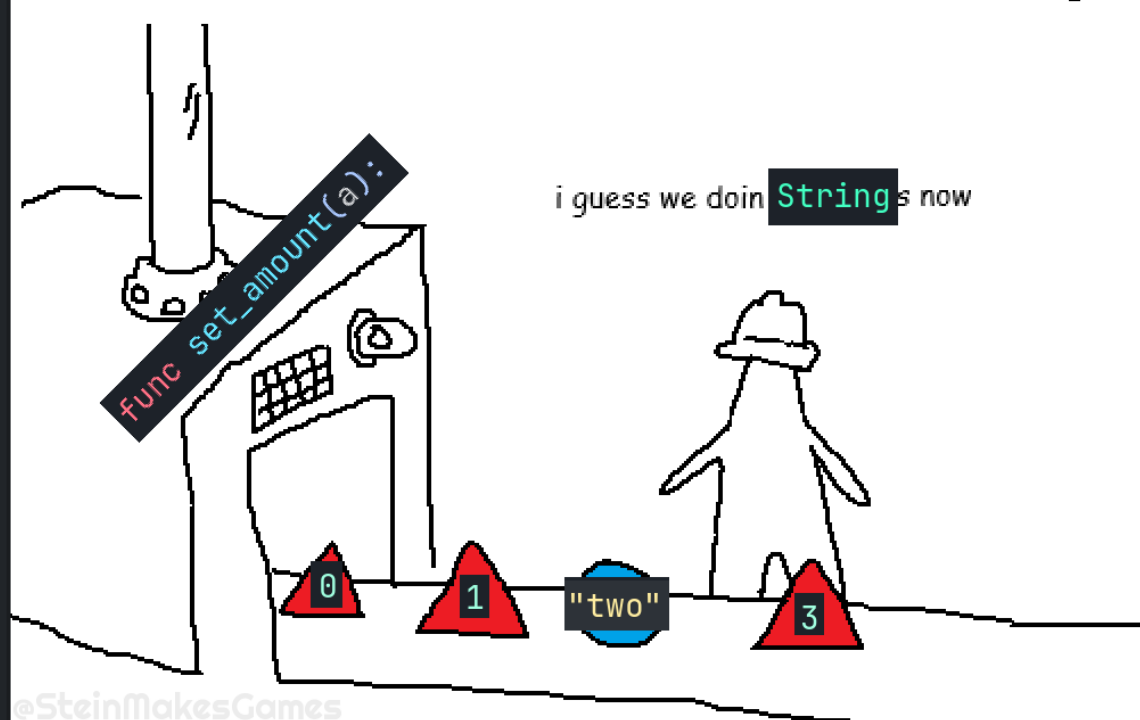
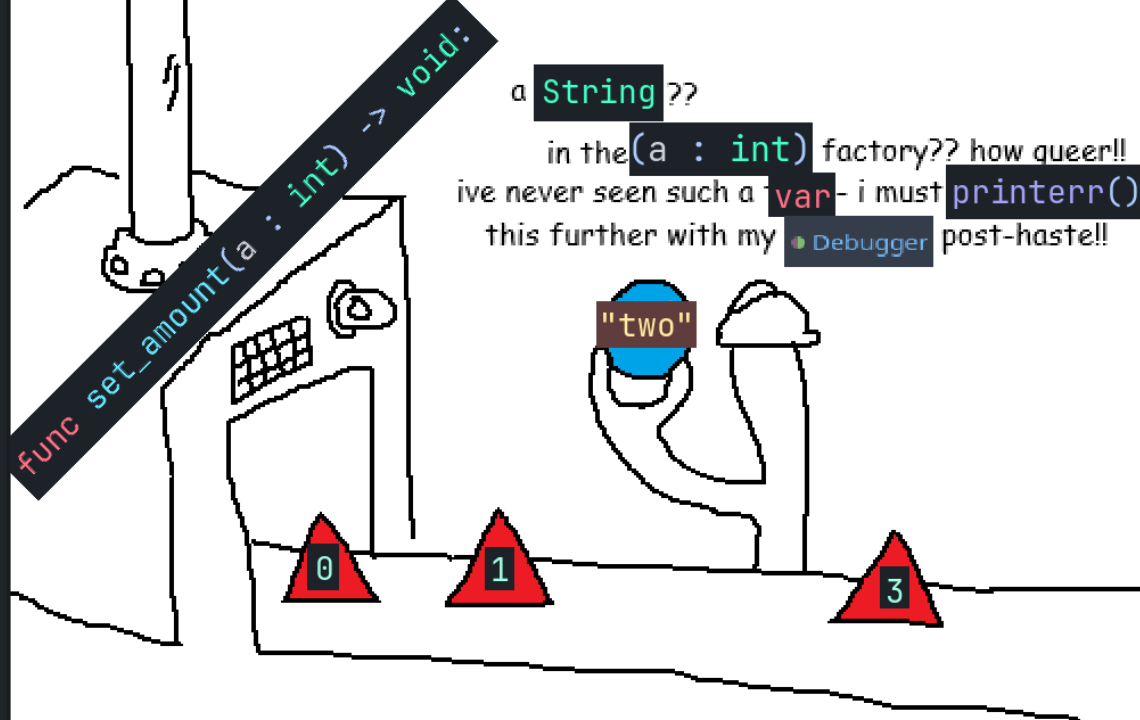


03 | Malware Linguistics

Choosing the right language for the job

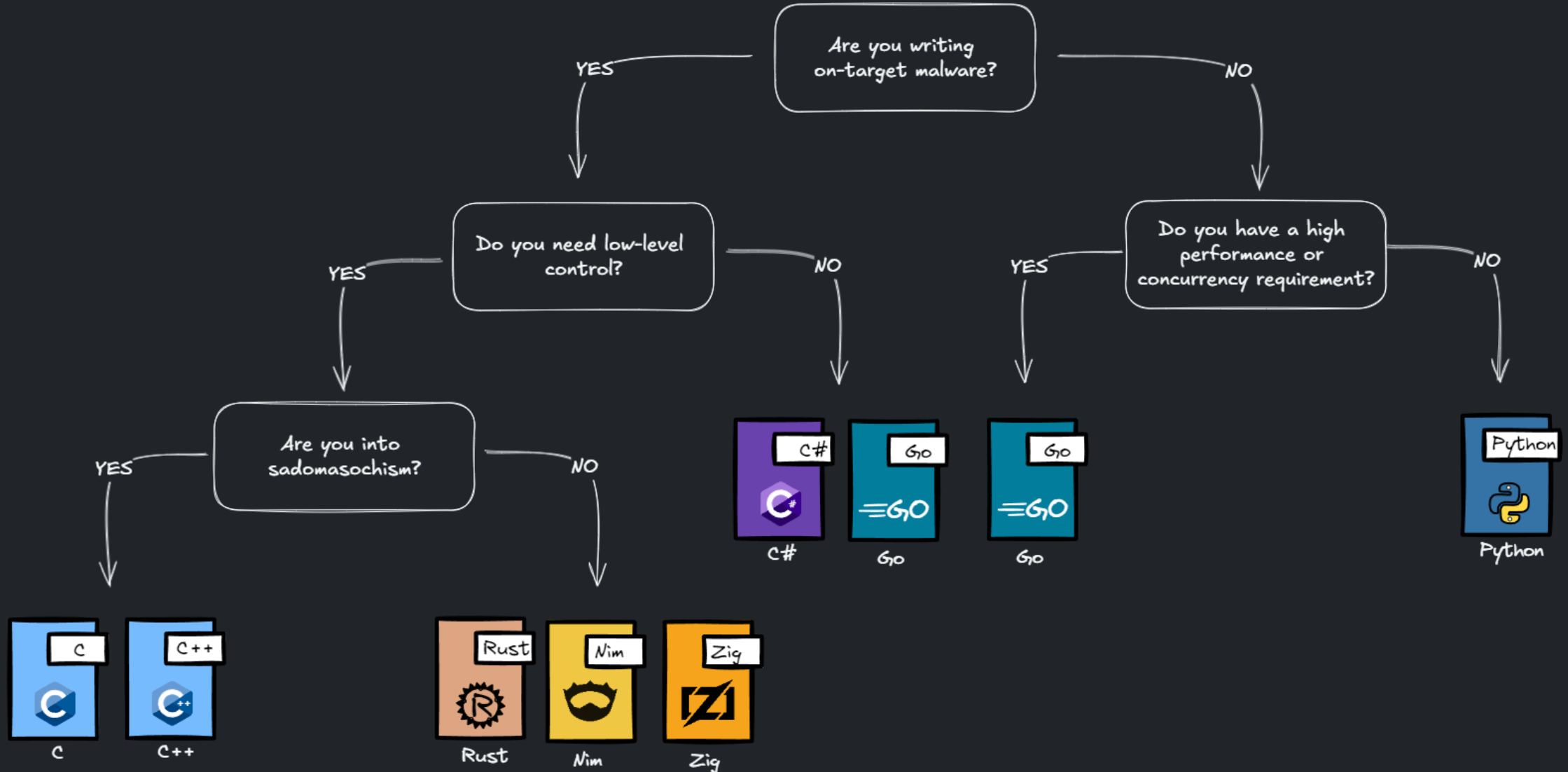
- Many programming languages can be used, each with benefits and drawbacks
- Considerations:
 - Interpreted or compiled
 - High or low level
 - Performance
 - Portability
 - Prevalence
 - Developer experience





03 | Malware Linguistics

A totally un-biased decision tree



03 | Malware Linguistics

Rust

Pros

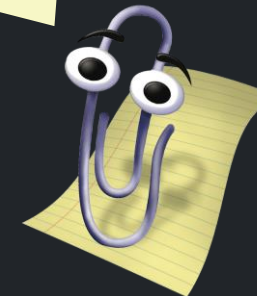
- Strongly typed, compiled
- “Borrow Checker”
 - No garbage collection
 - Compile-time safety checks
- Verbose, compile-time errors
- Mature “crate” ecosystem
- High adoption, blends in

Cons

- Steep learning curve
- Very verbose, hard to read
- Slow to compile
- Malware needs `unsafe{ }`

It looks like you're trying to write idiomatic Rust code!

Would you like to use `clippy::pedantic`?



Example OST

[FeroxBuster](#)
[NimPlant](#) (.rs)

Getting Started

[Rust by Example](#)
[The Rust Book](#)
[Black Hat Rust](#)

03 | Malware Linguistics

Rust

```
use windows::{
    core::*,
    Win32::UI::WindowsAndMessaging::{MessageBoxW, MB_OKCANCEL},
};

fn show_message() -> Result<Option<i32>> {
    let result = unsafe {
        MessageBoxW(
            None,
            w!("Click OK to continue, Cancel to quit"),
            w!("Message"),
            MB_OKCANCEL,
        )
    };

    match result.0 {
        0 => Err(Error::from_win32()),
        1 => Ok(Some(1)), // OK clicked
        2 => Ok(Some(2)), // Cancel clicked
        _ => Ok(None),   // Unexpected result
    }
}

fn main() -> Result<()> {
    let mut attempts = 3;
    while attempts > 0 {
        match show_message()? {
            Some(1) => println!("Continuing..."),
            Some(2) => break,
            _ => println!("Unexpected response"),
        }
        attempts -= 1;
    }
    Ok(())
}
```

03 | Malware Linguistics

Rust

```
use windows::{
    core::*,
    Win32::UI::WindowsAndMessaging::{MessageBoxW, MB_OKCANCEL},
};

fn show_message() -> Result<Option<i32>> {
    let result = unsafe {
        MessageBoxW(
            None,
            w!("Click OK to continue, Cancel to quit"),
            w!("Message"),
            MB_OKCANCEL,
        )
    };

    match result.0 {
        0 => Err(Error::from_win32()),
        1 => Ok(Some(1)), // OK clicked
        2 => Ok(Some(2)), // Cancel clicked
        _ => Ok(None), // Unexpected result
    }
}

fn main() -> Result<()> {
    let mut attempts = 3;
    while attempts > 0 {
        match show_message()? {
            Some(1) => println!("Continuing..."),
            Some(2) => break,
            _ => println!("Unexpected response"),
        }
        attempts -= 1;
    }
    Ok(())
}
```

```
use windows::{
    core::*,
    Win32::UI::WindowsAndMessaging::{MessageBoxW, MB_OKCANCEL},
};
```

03 | Malware Linguistics

Rust

```
use windows::{
    core::*,
    Win32::UI::WindowsAndMessaging::{MessageBoxW, MB_OKCANCEL},
};

fn show_message() -> Result<Option<i32>> {
    let result = unsafe {
        MessageBoxW(
            None,
            w!("Click OK to continue, Cancel to quit"),
            w!("Message"),
            MB_OKCANCEL,
        )
    };

    match result.0 {
        0 => Err(Error::from_win32()),
        1 => Ok(Some(1)), // OK clicked
        2 => Ok(Some(2)), // Cancel clicked
        _ => Ok(None), // Unexpected result
    }
}
```

```
fn main() -> Result<()> {
    let mut attempts = 3;
    while attempts > 0 {
        match show_message()? {
            Some(1) => println!("Continuing..."),
            Some(2) => break,
            _ => println!("Unexpected response"),
        }
        attempts -= 1;
    }
    Ok(())
}
```

```
fn main() -> Result<()> {
    let mut attempts = 3;
    while attempts > 0 {
        match show_message()? {
            Some(1) => println!("Continuing..."),
            Some(2) => break,
            _ => println!("Unexpected response"),
        }
        attempts -= 1;
    }
    Ok(())
}
```

03 | Malware Linguistics

Rust

```
use windows::{
    core::*,
    Win32::UI::WindowsAndMessaging::{MessageBoxW, MB_OKCANCEL},
};
```

```
fn show_message() -> Result<Option<i32>> {
    let result = unsafe {
        MessageBoxW(
            None,
            w!("Click OK to continue, Cancel to quit"),
            w!("Message"),
            MB_OKCANCEL,
        )
    };

    match result.0 {
        0 => Err(Error::from_win32()),
        1 => Ok(Some(1)), // OK clicked
        2 => Ok(Some(2)), // Cancel clicked
        _ => Ok(None),   // Unexpected result
    }
}
```

```
fn main() -> Result<()> {
    let mut attempts = 3;
    while attempts > 0 {
        match show_message()? {
            Some(1) => println!("Continuing..."),
            Some(2) => break,
            _ => println!("Unexpected response"),
        }
        attempts -= 1;
    }
    Ok(())
}
```

```
fn show_message() -> Result<Option<i32>> {
    let result = unsafe {
        MessageBoxW(
            None,
            w!("Click OK to continue, Cancel to quit"),
            w!("Message"),
            MB_OKCANCEL,
        )
    };
}
```

```
match result.0 {
    0 => Err(Error::from_win32()),
    1 => Ok(Some(1)), // OK clicked
    2 => Ok(Some(2)), // Cancel clicked
    _ => Ok(None),   // Unexpected result
}
```

```
}
```

03 | Malware Linguistics

Go

Pros

- Strongly typed, compiled
- Simple syntax, easy to learn
- Easy and performant concurrency
- Extensive standard library and package ecosystem
- Strong community and high prevalence

Cons

- Error handling can feel verbose
- Larger, identifiable binaries
- Garbage collected, less suitable for embedded platforms

Example OST

[Nuclei](#)
[Bettercap](#)

Getting Started

[Go by Example](#)
[Effective Go](#)
[Black Hat Go](#)



03 | Malware Linguistics

Go

```
package main

import (
    "fmt"
    "net"
    "sync"
    "time"
)

func scanPort(host string, port int, wg *sync.WaitGroup) {
    defer wg.Done()

    address := fmt.Sprintf("%s:%d", host, port)
    conn, err := net.DialTimeout("tcp", address, 1*time.Second)
    if err == nil {
        conn.Close()
        fmt.Printf("Port %d is open\n", port)
    }
}

func main() {
    host := "example.com"
    var wg sync.WaitGroup

    for port := 1; port <= 1024; port++ {
        wg.Add(1)
        go scanPort(host, port, &wg)
    }

    wg.Wait()
    fmt.Println("Port scan completed")
}
```

03 | Malware Linguistics

Go



```
package main
```

```
import (  
    "fmt"  
    "net"  
    "sync"  
    "time"  
)
```

```
func scanPort(host string, port int, wg *sync.WaitGroup) {  
    defer wg.Done()  
  
    address := fmt.Sprintf("%s:%d", host, port)  
    conn, err := net.DialTimeout("tcp", address, 1*time.Second)  
    if err == nil {  
        conn.Close()  
        fmt.Printf("Port %d is open\n", port)  
    }  
}  
  
func main() {  
    host := "example.com"  
    var wg sync.WaitGroup  
  
    for port := 1; port <= 1024; port++ {  
        wg.Add(1)  
        go scanPort(host, port, &wg)  
    }  
  
    wg.Wait()  
    fmt.Println("Port scan completed")  
}
```

```
import (  
    "fmt"  
    "net"  
    "sync"  
    "time"  
)
```

03 | Malware Linguistics

Go

```
package main

import (
    "fmt"
    "net"
    "sync"
    "time"
)

func scanPort(host string, port int, wg *sync.WaitGroup) {
    defer wg.Done()

    address := fmt.Sprintf("%s:%d", host, port)
    conn, err := net.DialTimeout("tcp", address, 1*time.Second)
    if err == nil {
        conn.Close()
        fmt.Printf("Port %d is open\n", port)
    }
}
```

```
func main() {
    host := "example.com"
    var wg sync.WaitGroup

    for port := 1; port <= 1024; port++ {
        wg.Add(1)
        go scanPort(host, port, &wg)
    }

    wg.Wait()
    fmt.Println("Port scan completed")
}
```

```
func main() {
    host := "example.com"
    var wg sync.WaitGroup

    for port := 1; port <= 1024; port++ {
        wg.Add(1)
        go scanPort(host, port, &wg)
    }

    wg.Wait()
    fmt.Println("Port scan completed")
}
```

03 | Malware Linguistics

Go

```
package main

import (
    "fmt"
    "net"
    "sync"
    "time"
)

func scanPort(host string, port int, wg *sync.WaitGroup) {
    defer wg.Done()

    address := fmt.Sprintf("%s:%d", host, port)
    conn, err := net.DialTimeout("tcp", address, 1*time.Second)
    if err == nil {
        conn.Close()
        fmt.Printf("Port %d is open\n", port)
    }
}

func main() {
    host := "example.com"
    var wg sync.WaitGroup

    for port := 1; port <= 1024; port++ {
        wg.Add(1)
        go scanPort(host, port, &wg)
    }

    wg.Wait()
    fmt.Println("Port scan completed")
}
```

```
func scanPort(host string, port int, wg *sync.WaitGroup) {
    defer wg.Done()

    address := fmt.Sprintf("%s:%d", host, port)
    conn, err := net.DialTimeout("tcp", address, 1*time.Second)
    if err == nil {
        conn.Close()
        fmt.Printf("Port %d is open\n", port)
    }
}
```

04 | Takeaways

Go forth and develop offensively!



Build your own tools

...or hire someone to do it.
It's no longer optional



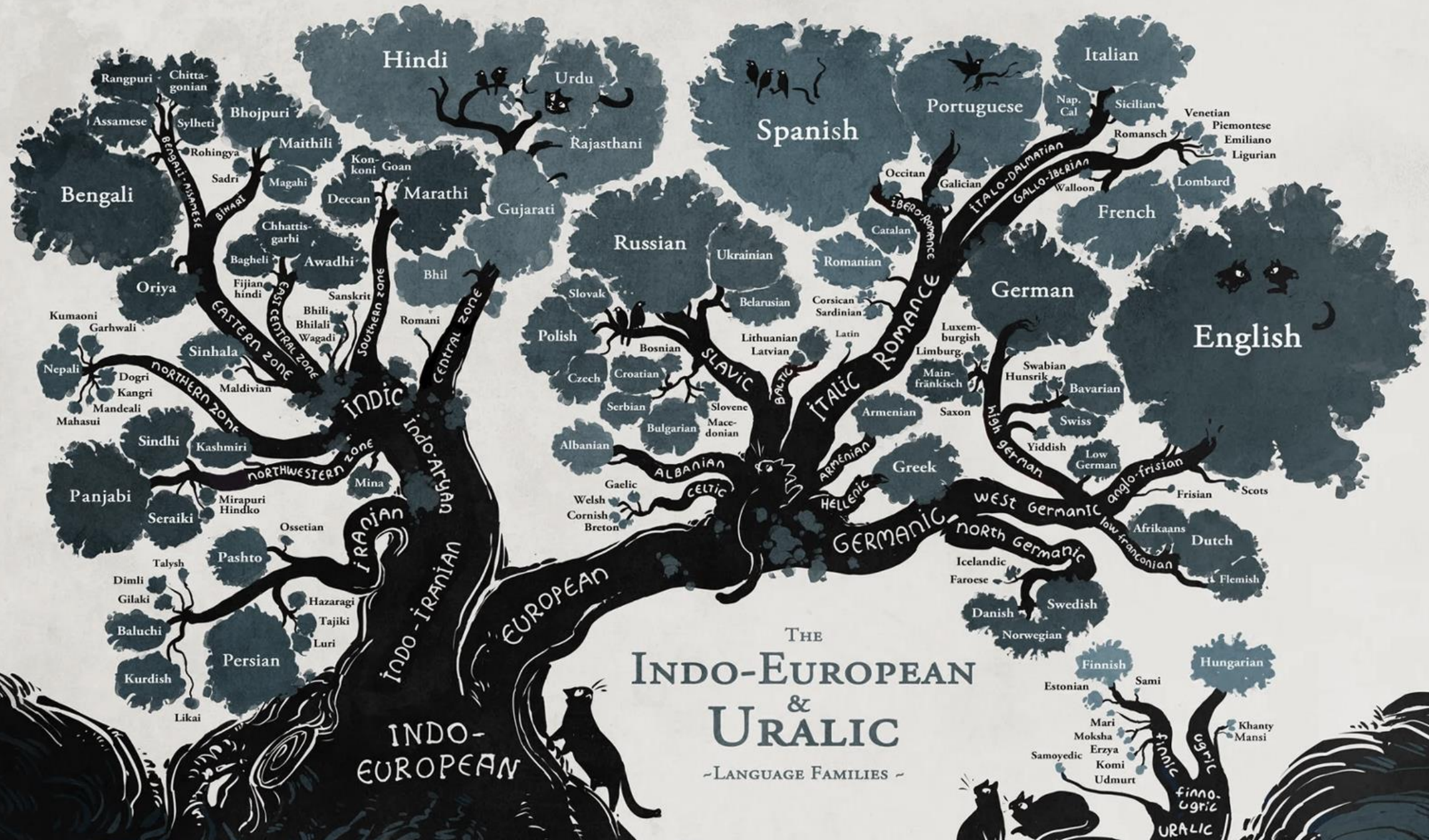
Be a good dev

Strive for readable and
maintainable code to
improve collaboration



Learn all the languages

Try many languages and
see which ones work for
you (and your project)



THE
**INDO-EUROPEAN
 &
 URALIC**
 ~LANGUAGE FAMILIES~

Graphic: Minna Sundberg