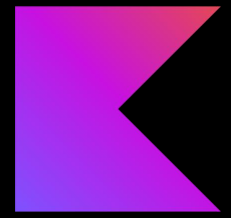



Popular vs Less Well Known Programming Languages



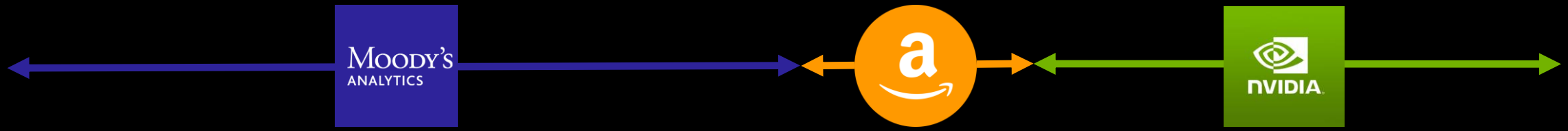



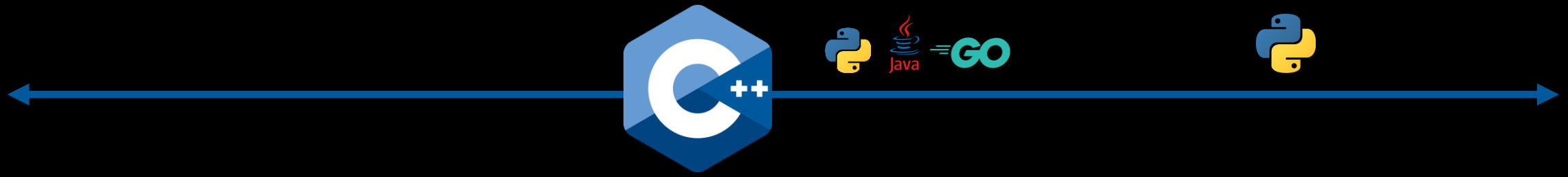
https://github.com/codereport/Content

Conor Hoekstra

code_report

codereport





About Me

Conor Hoekstra / @code_report

C++ vs CUDA vs APL vs BQN

12K views • 11 months ago

Functional vs Array Programming

81K views • 11 months ago

Four More APL Solutions in 10 Minutes!

4.1K views • 11 months ago

4 APL Solutions in 10 Minutes!

4.8K views • 1 year ago

1 Problem, 5 Programming Languages

13K views • 1 year ago

1 Problem, 16 Programming Languages (C++ vs Rust vs...

134K views • 1 year ago

APL-Inspired Python!

12K views • 1 year ago

Python vs 3 Character APL Solution

5K views • 1 year ago

Python vs APL (1 Problem)

13K views • 1 year ago

One Problem, Five Programming Languages...

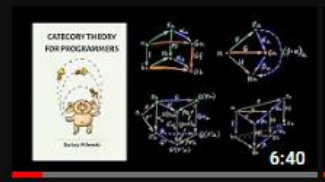
3.7K views • Streamed 1 year ago

APL + Game of Life = ❤

23K views • 1 year ago

Category Theory for Programmers: Chapter 11 -...

1.8K views • 1 year ago

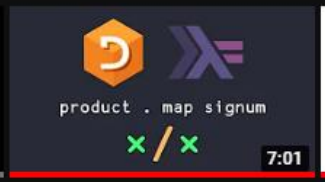


Category Theory for Programmers: Chapter 10 -...

2.9K views • 1 year ago

Thrust and the C++ Standard Algorithms - Conor Hoekstr...

3.3K views • 1 year ago

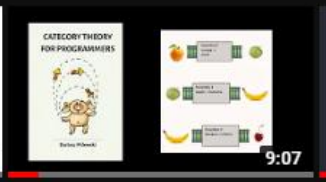


I ❤ APL and Haskell #2

8.2K views • 1 year ago

1 Problem, 4 More Programming Languages...

41K views • 1 year ago



Category Theory for Programmers: Chapter 9 -...

1.6K views • 1 year ago

I ❤ APL and Haskell

17K views • 1 year ago

APL Wins (vs C++, Java & Python)

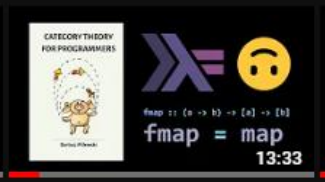
16K views • 1 year ago

Category Theory for Programmers: Chapter 8 -...

2.9K views • 1 year ago

J Language: From C to C++20 - LiveStream #12 (Code...

871 views • Streamed 1 year ago



Category Theory for Programmers: Chapter 7 -...

2.8K views • 1 year ago

Category Theory for Programmers: Chapter 6 -...

3.1K views • 1 year ago

Category Theory for Programmers: Chapter 5 -...

3.9K views • 1 year ago




Popular vs Less Well Known
Programming Languages






Popular




Less Well Known

Language Rankings

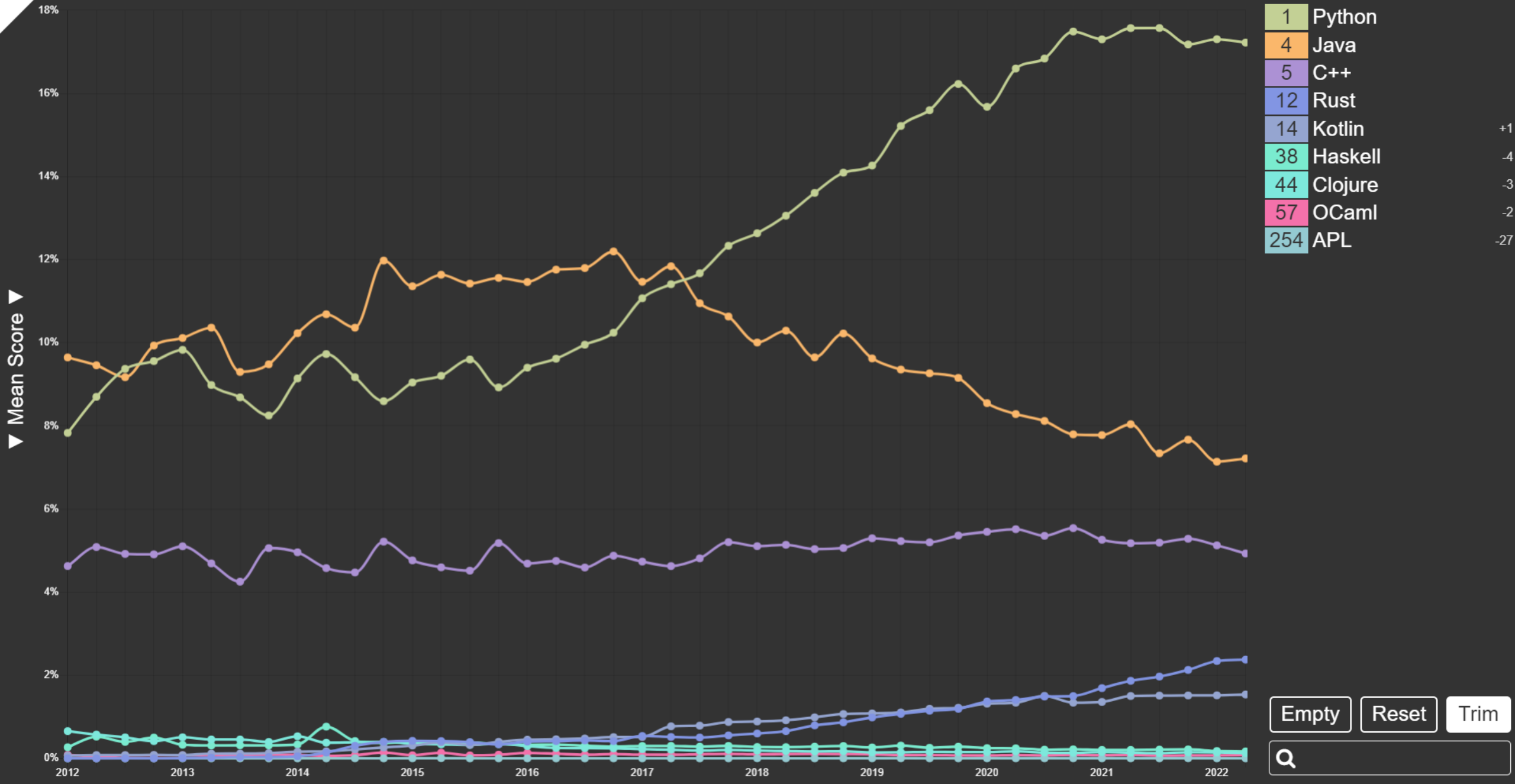
https://www.tiobe.com/tiobe-index/
https://pypl.github.io/PYPL.html
https://insights.stackoverflow.com/survey/
https://redmonk.com/sogrady/2022/03/28/language-rankings-1-22/
https://spectrum.ieee.org/top-programming-languages-2022
https://octoverse.github.com/#top-languages-over-the-years
https://madnight.github.io/githut/#/pull_requests/2022/1

Languish

https://tjpalmer.github.io/languish/

Languish

Programming Language Trends
... for more, subscribe to Context Free



1. 
2. 

3. 
4. 
5. 

Filtering Odd Numbers
(aka keeping odd numbers)

Filtering Odd Numbers

```
[1,2,3,4,5]
```

Filtering Odd Numbers

```
filter_odds([1,2,3,4,5])
```

Filtering Odd Numbers

```
filter_odds([1,2,3,4,5])
```

```
[1,3,5]
```

```python
def filter_odds(lst):
    res = []
    for e in lst:
        if e % 2:
            res.append(e)
    return res
```

```cpp
auto filter_evens(std::vector<int> list) {
  auto res = std::vector<int>{};
  for (auto num : list) {
    if (num % 2) {
      res.push_back(num);
    }
  }
  return res;
}
```


```java
List<Integer> filterOdds(List<Integer> list) {
    var res = new ArrayList<Integer>();
    for (var e : list) {
        if (e % 2 == 1) {
            res.add(e);
        }
    }
    return res;
}
```

```
filterOdds xs = filter odd xs
```

```
filterOdds = filter odd
```

```
filterOdds = filter (\e -> (mod e 2) == 1)
```

```
filterOdds = filter odd
```

```
filterOdds xs = [ x | x <- xs, odd x ]
```

```haskell
filterOdds :: [Integer] -> [Integer]
filterOdds = filter odd
```

Hoogle

[Integer] -> [Integer]

set:stackage

Search

Packages

- is:exact
- base
- ghc
- hedgehog
- ihaskell
- Cabal
- safe
- protolude
- extra
- MissingH
- base-prelude
- rio
- numeric-prelude
- relude
- universum
- classy-prelude
- basement

:: [Integer] -> [Integer]

tail :: HasCallStack => [a] -> [a]

base Prelude Data.List GHC.List GHC.OldList, ghc GHC.Prelude

Extract the elements after the head of a list, which must be non-empty.

init :: HasCallStack => [a] -> [a]

base Prelude Data.List GHC.List GHC.OldList, ghc GHC.Prelude

Return all the elements of a list except the last one. The list must be non-empty.

reverse :: [a] -> [a]

base Prelude Data.List GHC.List GHC.OldList, ghc GHC.Prelude, hedgehog Hedgehog.Internal.Prelude, ihaskell IHaskellPrelude

reverse xs returns the elements of xs in reverse order. xs must be finite.

cycle :: HasCallStack => [a] -> [a]

base Prelude Data.List GHC.List GHC.OldList, ghc GHC.Prelude

cycle ties a finite list into a circular one, or equivalently, the infinite repetition of the original list. It is the identity on infinite lists.

safeTail :: [a] -> [a]

Cabal Distribution.Simple.Utils

A total variant of tail.

safeInit :: [a] -> [a]

Cabal Distribution.Simple.Utils

```
let filterOdds l =
    List.filter (fun x -> x mod 2 != 0) l ;;
```

```clojure
(defn filter-odds [list]
  (filter odd? list))
```

```clojure
(defn filter-odds1 [list]
    (filter #(= 1 (mod % 2)) list))
```

```clojure
(defn filter-odds [list]
  (filter odd? list))
```

```clojure
(defn sum-first-two-odds [list]
    (reduce + (take 2 (filter odd? list))))
```

```clojure
(defn sum-first-two-odds [list]
      (->> list
           (filter odd?)
           (take 2)
           (reduce +)))
```

```
FilterOdds[list_] := Select[list, OddQ]
```


```
filterOdds ← {(2|⍵)/⍵}
```

$2 \mid \omega$

2 | 1 2 3 4 5

2 | 1 2 3 4 5

1 0 1 0 1

1 0 1 0 1/1 2 3 4 5

1 0 1 0 1

1 2 3 4 5

1 0 1 0 1

1 2 3 4 5

1 3 5


```
{(2|⍵)/⍵}
```


```
{(2|⍵)/⍵}
{⍵/⍨2|⍵}
```


```
{(2|⍵)/⍵}
{⍵/⍨2|⍵}
(2|⊢)⊢⌸/⊢
```

```
{(2|⍵)/⍵}
 {⍵/⍨2|⍵}
(2|⊢)⊢⌸/⊢
```

```
{(2|⍵)/⍵}
 {⍵/⍨2|⍵}
(2|⊢)⊢⌸/⊢
```

```
{(2|𝕩)/𝕩}
```

```
{(2|⍵)/⍵}
{⍵/⍨2|⍵}
(2|⊢)⊢⌸/⊢
```

```
{(2|𝕩)/𝕩}
{𝕩/˜2|𝕩}
```

```
{(2|⍵)/⍵}
{⍵/⍨2|⍵}
(2|⊢)⊢⌸/⊢
```

```
{(2|𝕩)/𝕩}
{𝕩/˜2|𝕩}
(2|⊢)/⊢
```

```
{(2|⍵)/⍵}
{⍵/⍨2|⍵}
(2|⊢)⊢⌸/⊢
```

```
{(2|𝕩)/𝕩}
{𝕩/˜2|𝕩}
(2|⊢)/⊢
⊢/˜2|⊢
```




Popular




Less Well Known

```python
def filter_odds(lst):
    res = []
    for e in lst:
        if e % 2:
            res.append(e)
    return res
```

```python
def filter_odds(lst):
    return list(filter(lambda e: e % 2, lst))
```

```python
def filter_odds(lst):
    return list(filter(lambda e: e % 2, lst))
```

```python
def filter_odds(lst):
    return [e for e in lst if e % 2]
```

```cpp
auto filter_evens(std::vector<int> list) {
  auto res = std::vector<int>{};
  for (auto num : list) {
    if (num % 2) {
      res.push_back(num);
    }
  }
  return res;
}
```

```cpp
auto filter_evens(std::vector<int> list) {
  return list
    | std::views::filter(
        [](auto e) { return e % 2; });
}
```

```cpp
auto filter_evens(std::vector<int> list) {
  auto odd = [](auto e) { return e % 2; };
  return list | std::views::filter(odd);
}
```

```cpp
auto filter_evens(std::vector<int> list) {
  auto odd = [](auto e) { return e % 2; };
  return list | std::views::filter(odd)
              | std::views::take(2)
              | // ...
}
```

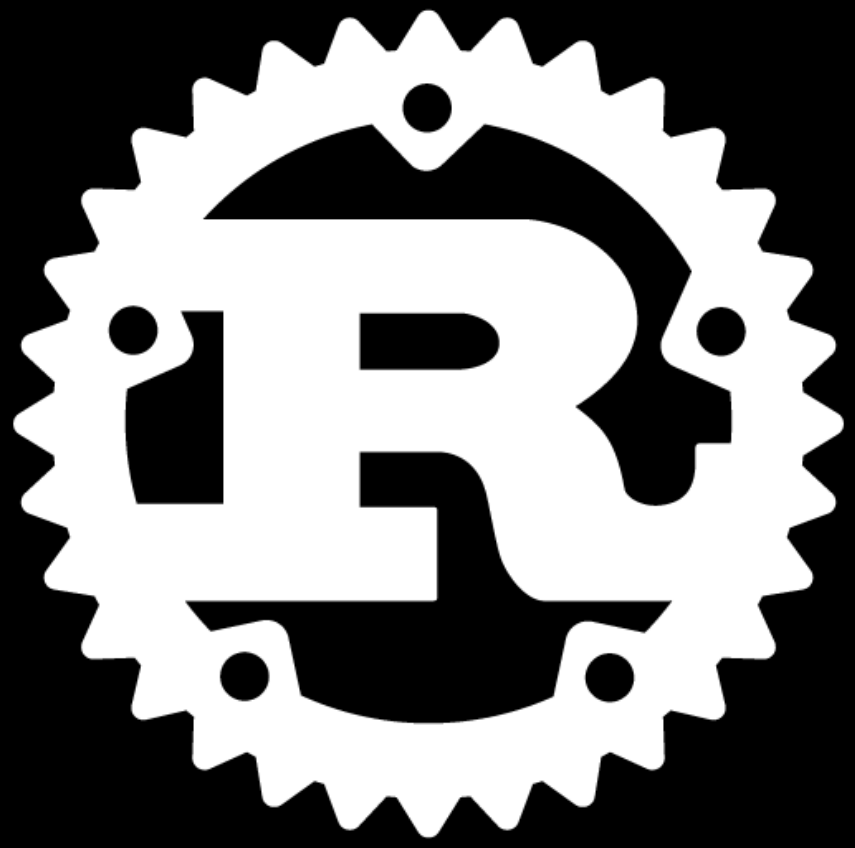
```cpp
auto filter_evens(std::vector<int> list) {
  using namespace std::views;
  auto odd = [](auto e) { return e % 2; };
  return list | filter(odd)
              | take(2)
              | // ...
}
```

```
auto filter_evens(std::vector<int> list) {
  using namespace std::views;
  auto odd = [](auto e) { return e % 2; };

  return list | filter(odd)
              | take(2)
              | // ...

}
```


```java
List<Integer> filterOdds(List<Integer> list) {
    var res = new ArrayList<Integer>();
    for (var e : list) {
        if (e % 2 == 1) {
            res.add(e);
        }
    }
    return res;
}
```


```java
List<Integer> filterOdds(List<Integer> list) {
    return list.stream()
               .filter(e -> e % 2 == 1)
               .collect(Collectors.toList());
}
```


```
Integer sumFirstTwoOdds(List<Integer> list) {
    return list.stream()
               .filter(e -> e % 2 == 1)
               .limit(2)
               .reduce(0, Integer::sum);
}
```



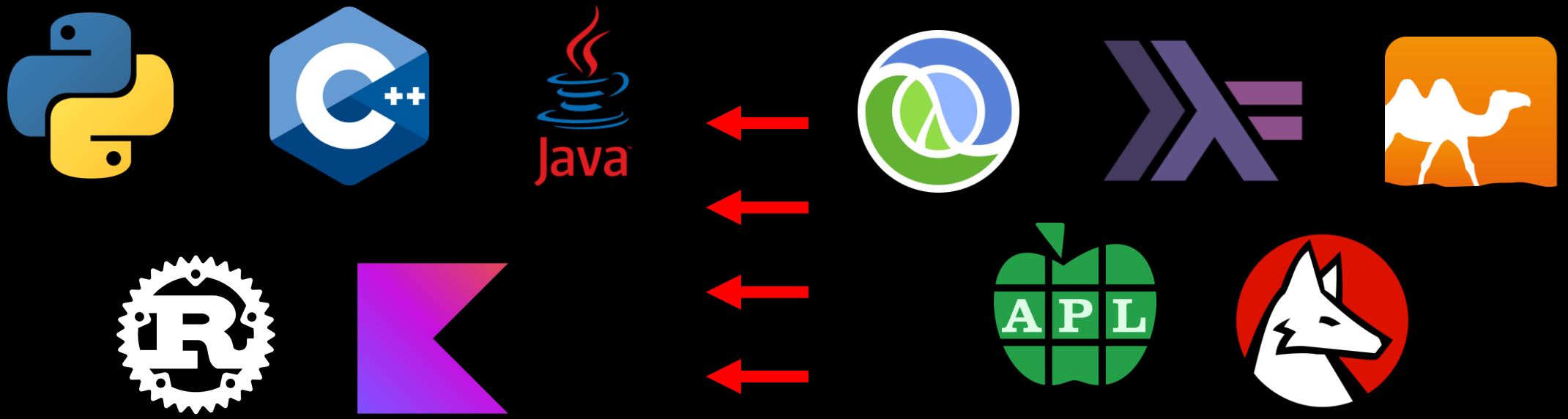
```rust
fn filter_odds(list: Vec<i32>) -> Vec<i32> {
    list.into_iter()
        .filter(|e| e % 2 == 1)
        .collect()
}
```

```rust
fn sum_first_two_odds(list: Vec<i32>) -> i32 {
    list.into_iter()
        .filter(|e| e % 2 == 1)
        .take(2)
        .sum()
}
```

```
fun sumFirstTwoOdds(lst: List<Int>): Int
    = lst.filter{ x -> x % 2 == 1 }
         .take(2)
         .sum()
```

```kotlin
fun sumFirstTwoOdds(lst: List<Int>): Int
    = lst.filter{ x -> x % 2 == 1 }
         .take(2)
         .sum()
```

```rust
fn sum_first_two_odds(list: Vec<i32>) -> i32 {
    list.into_iter()
        .filter(|e| e % 2 == 1)
        .take(2)
        .sum()
}
```



Popular

Less Well Known

```
{2|⍵}
```

```
|e| e % 2 == 1
```

```
e -> e % 2 == 1
```

```
e -> e % 2 == 1
```

```
#(= 1 (mod % 2))
```

```
lambda e: e % 2
```

```
fun e -> e mod 2 != 0
```

```
(\e -> (mod e 2) == 1)
```

```
Function[{x}, Mod[x,2]]
```

```
[](auto e) { return e % 2; }
```

```
{2|ω}
```

```
|e| e % 2 == 1
```

```
e -> e % 2 == 1
```

```
e -> e % 2 == 1
```

```
#(= 1 (mod % 2))
```

```
lambda e: e % 2
```

```
fun e -> e mod 2 != 0
```

```
(\e -> (mod e 2) == 1)
```

```
Function[{x}, Mod[x,2]]
```

```
[](auto e) { return e % 2; }
```

```
{2|ω}
```

```
|e| e % 2 == 1
```

```
e -> e % 2 == 1
```

```
e -> e % 2 == 1
```

```
#(= 1 (mod % 2))
```

```
lambda e: e % 2 == 1
```

```
fun e -> e mod 2 != 0
```

```
(\e -> (mod e 2) == 1)
```

```
Function[{x}, Mod[x,2]]
```

```
[](auto e) { return e % 2 == 1; }
```

```
(<0)                                  // 4:  Haskell
_ < 0                                 // 5:  Scala
_1 < 0                                // 6:  Boost.Lambda
#(< % 0)                              // 8:  Clojure
&(&1 < 0)                             // 9:  Elixir
|e| e < 0                             // 9:  Rust
\(e) e < 0                            // 10: R 4.1
{ $0 < 0 }                            // 10: Swift
{ it < 0 }                            // 10: Kotlin
e -> e < 0                            // 10: Java
e => e < 0                            // 10: C#, JS, Scala
\e -> e < 0                           // 11: Haskell
{ |e| e < 0 }                         // 13: Ruby
{ e in e < 0 }                        // 14: Swift
{ e -> e < 0 }                        // 14: Kotlin
fun e -> e < 0                        // 14: F#, OCaml
lambda e: e < 0                       // 15: Python
(λ (x) (< x 0))                       // 15: Racket
fn x -> x < 0 end                     // 17: Elixir
(lambda (x) (< x 0))                  // 20: Racket/Scheme/LISP
[](auto e) { return e < 0; }          // 28: C++
std::bind(std::less{}, _1, 0)         // 29: C++
func(e int) bool { return e < 0 }     // 33: Go
```

https://brevzin.github.io/c++/2020/06/18/lambda-lambda-lambda/

Thank you!

https://github.com/codereport/Content

Conor Hoekstra

code_report

codereport

Questions?

https://github.com/codereport/Content

Conor Hoekstra

code_report

codereport