To $\overset{\text{Code}}{\vee}$ Search, or Not to $\overset{\text{Code}}{\vee}$ Search

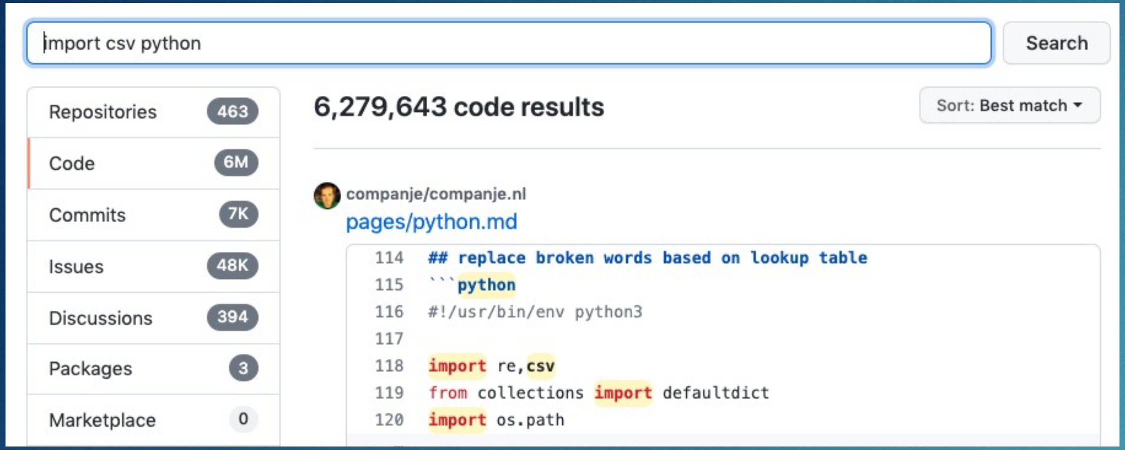
DEPENDS ON THE QUESTION

DR. KATIE STOLEE

ASSOCIATE PROFESSOR

NC STATE UNIVERSITY

Many interfaces.

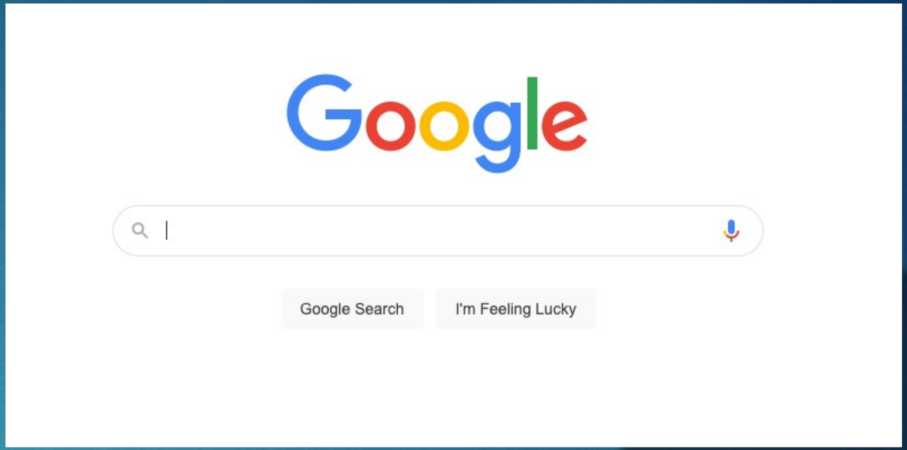


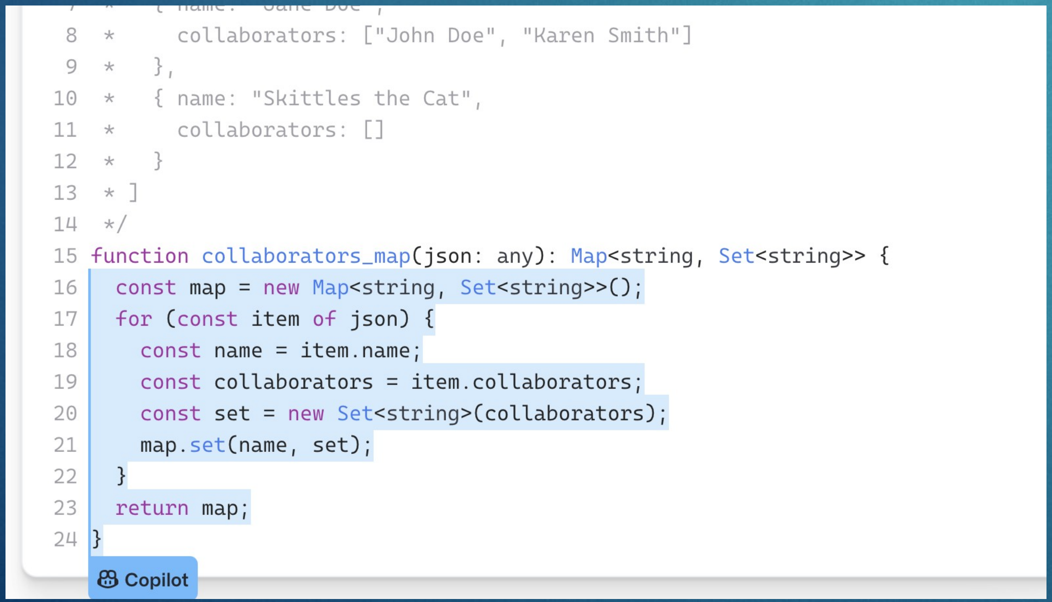
Repositories	463
Code	6M
Commits	7K
Issues	48K
Discussions	394
Packages	3
Marketplace	0

6,279,643 code results

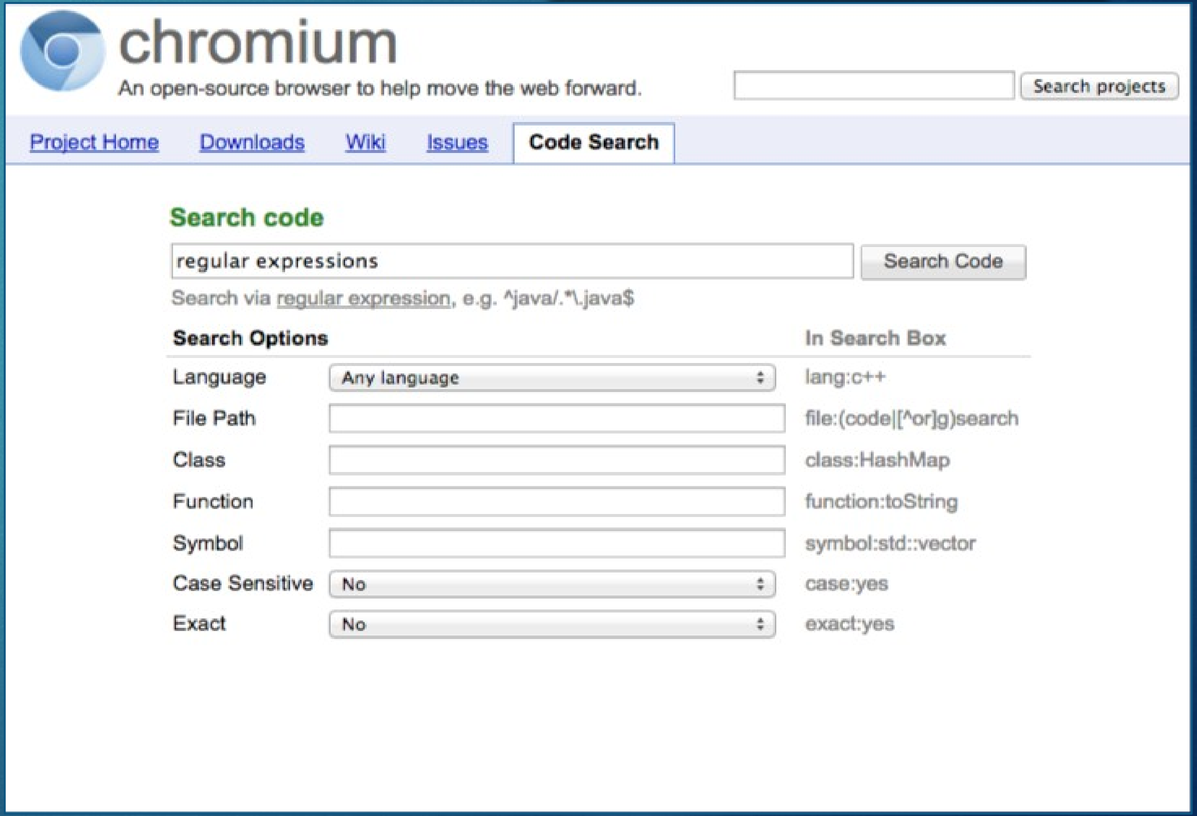
companje/companje.nl
pages/python.md

~~~
## replace broken words based on lookup table
```python
#!/usr/bin/env python3

import re,csv
from collections import defaultdict
import os.path
~~~





```
 *     collaborators: ["John Doe", "Karen Smith"]
 *   },
 *   { name: "Skittles the Cat",
 *     collaborators: []
 *   }
 * ]
 */
function collaborators_map(json: any): Map<string, Set<string>> {
  const map = new Map<string, Set<string>>();
  for (const item of json) {
    const name = item.name;
    const collaborators = item.collaborators;
    const set = new Set<string>(collaborators);
    map.set(name, set);
  }
  return map;
}
```



An open-source browser to help move the web forward.

Search code

regular expressions

Search via regular expression, e.g. ^java/.*\.java$

Search Options		In Search Box
Language	Any language	lang:c++
File Path		file:(code|[^or]g)search
Class		class:HashMap
Function		function:toString
Symbol		symbol:std::vector
Case Sensitive	No	case:yes
Exact	No	exact:yes

Code search is frequent

- ~12x per developer per day
- Search sessions involve multiple queries
- Code search with Google takes more time, more clicks, and more query reformulation than non-code search

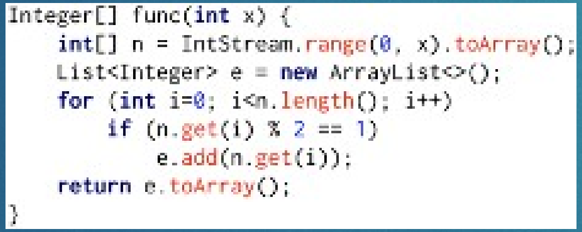
Four Distinct Needs

1. Example Code, **how** to do something (33%)
2. Explaining **what** it does (26%)
3. **Where** in the code base (16%)
4. **Why** is the code doing something (16%)

“How” → Example Code

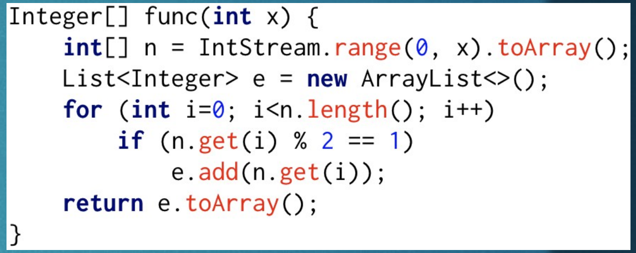
I have…

Java **for** loop to populate array of *even* numbers

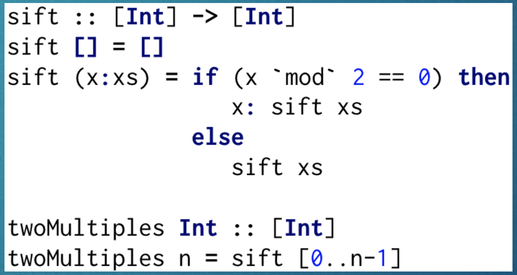


```
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

I want…

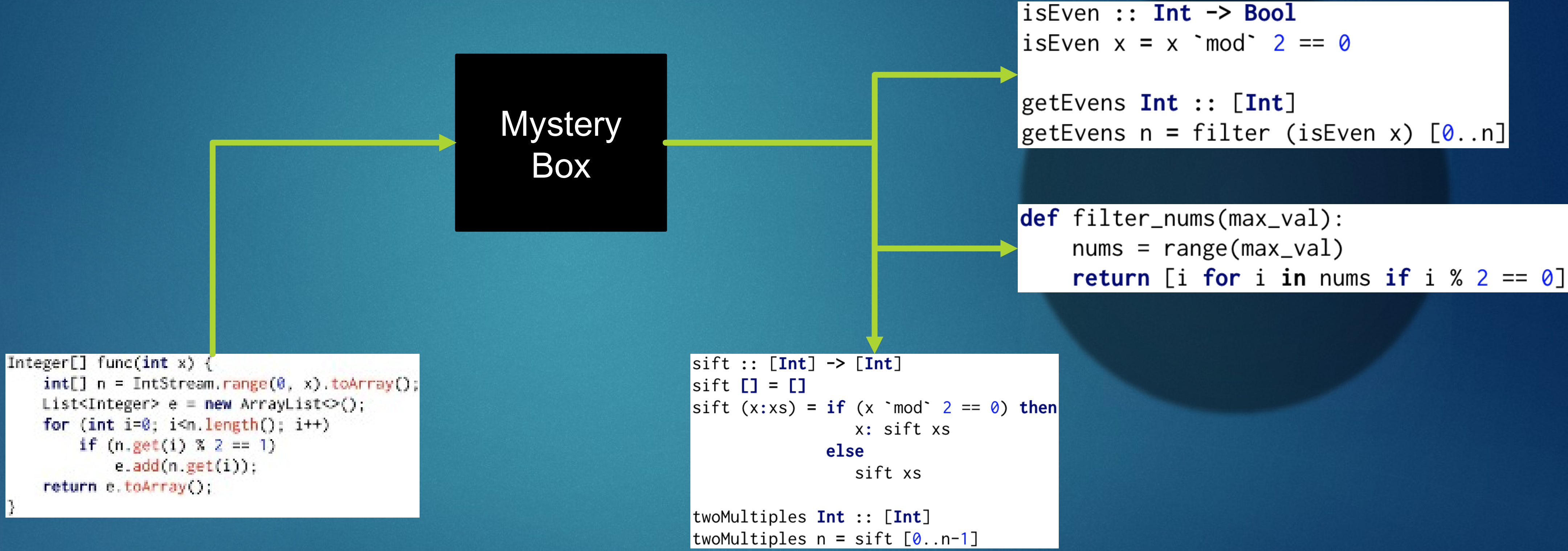


```
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```



```
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                 x: sift xs
              else
                 sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Code-to-Code Search



```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

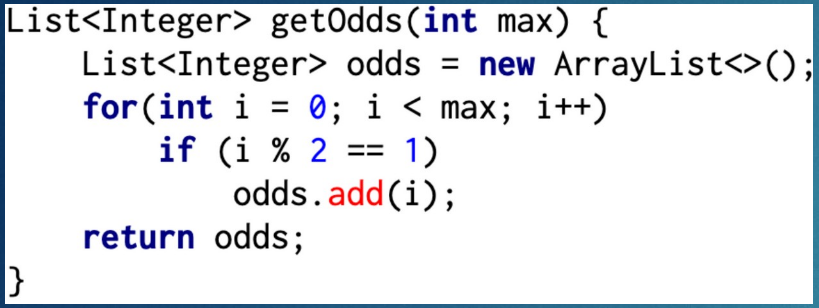
```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                 x: sift xs
              else
                 sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

The Halting Problem

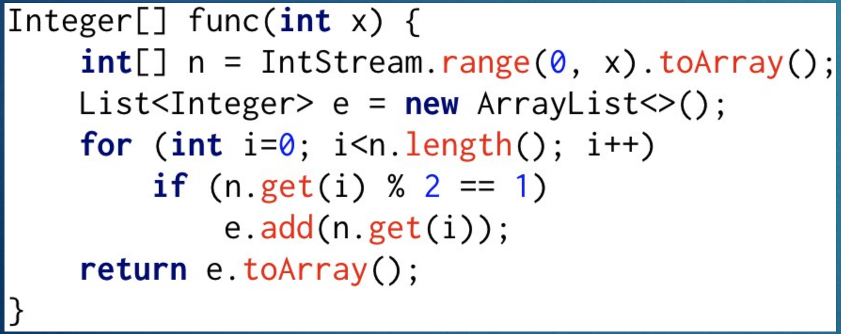
IT’LL NEVER WORK IN THEORY.

Code-to-code Search



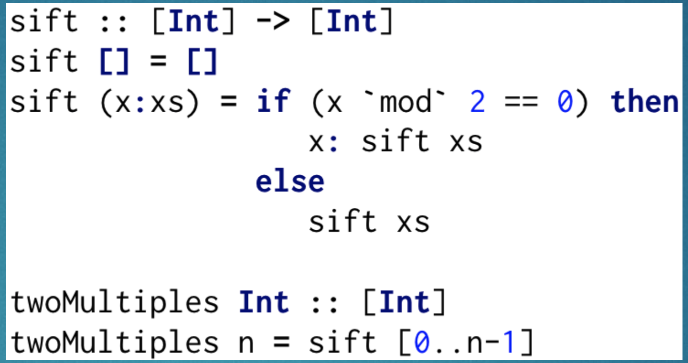
```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers



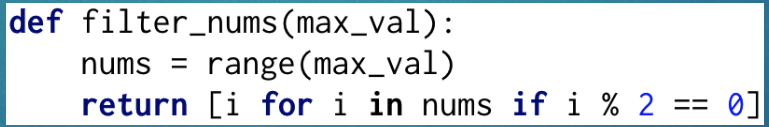
```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**



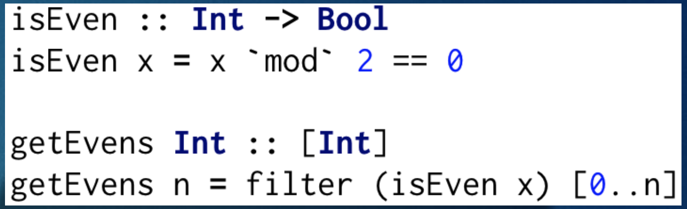
```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                  x: sift xs
              else
                  sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion



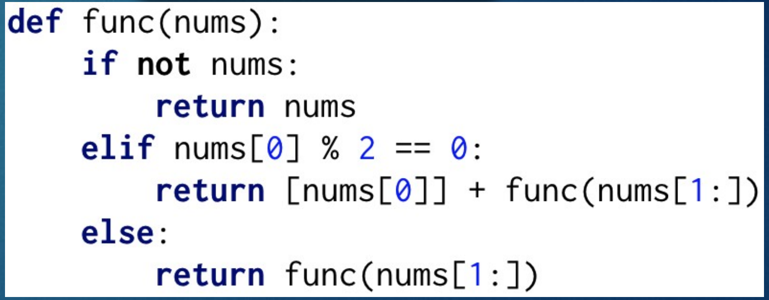
```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension



```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining



```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - Language

```
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

```
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                  x: sift xs
              else
                  sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

```
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - Behavior

```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                x: sift xs
            else
                sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - Structure

```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                    x: sift xs
                else
                    sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

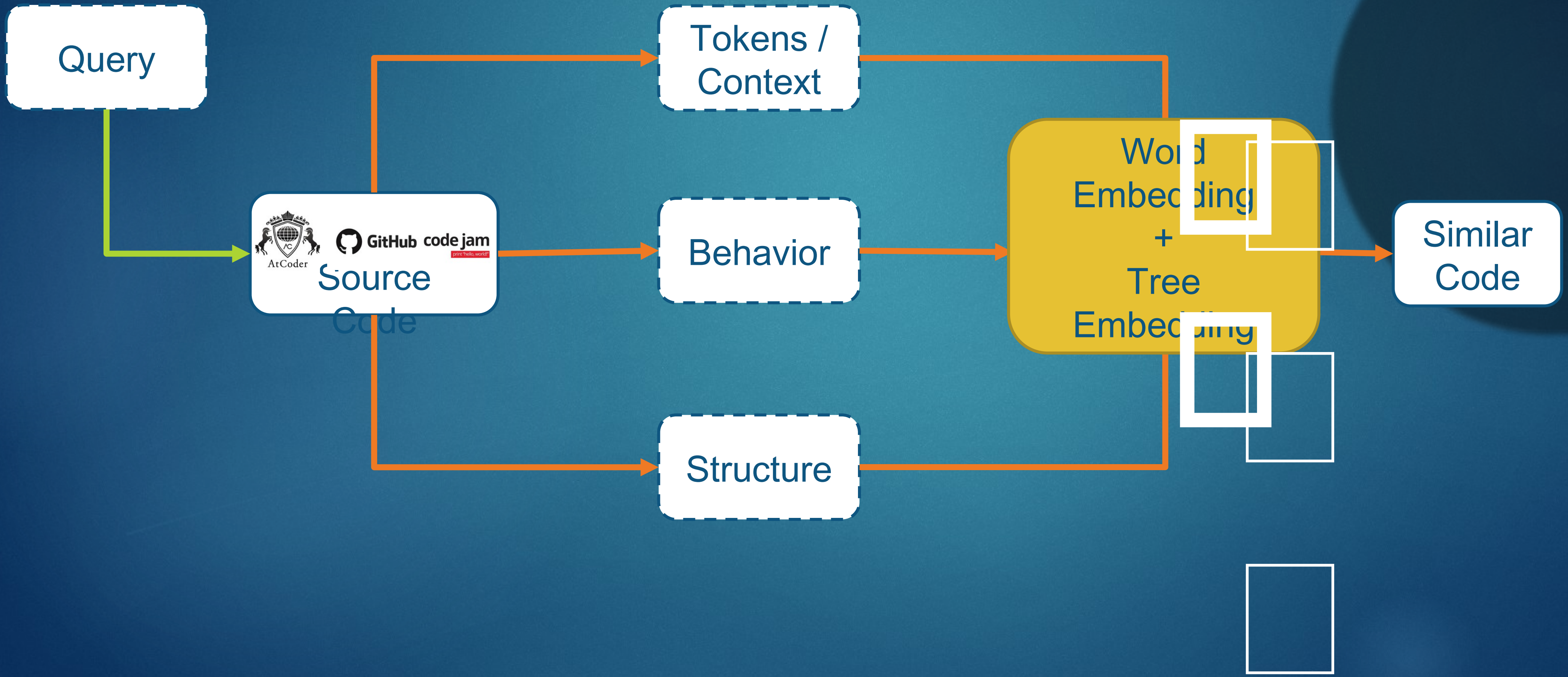
```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - In Practice



Four Distinct Needs

1. Example Code, **how** (33%)
2. Explaining **what** it does (26%)
3. **Where** in the code base (16%)
4. **Why** is the code doing something (16%)

Four Distinct Needs

1. Example Code, **how** (33%)

   Can be done in practice with search

2. Explaining **what** it does (26%)

   Code comprehension – not search

3. **Where** in the code base (16%)

   Code Navigation – works pretty well

4. **Why** is the code doing something (16%)

   Impact analysis – not search

How?

What? □ ☎

□

Where?

Why? □

□

Know **why** you're searching!

Thank you for listening.

KTSTOLEE@NCSU.EDU

Thank you to my collaborators:

- Sebastian Elbaum
- George Mathew
- Baishakhi Ray
- Caitlin Sadowski

Thank you to my sponsors: