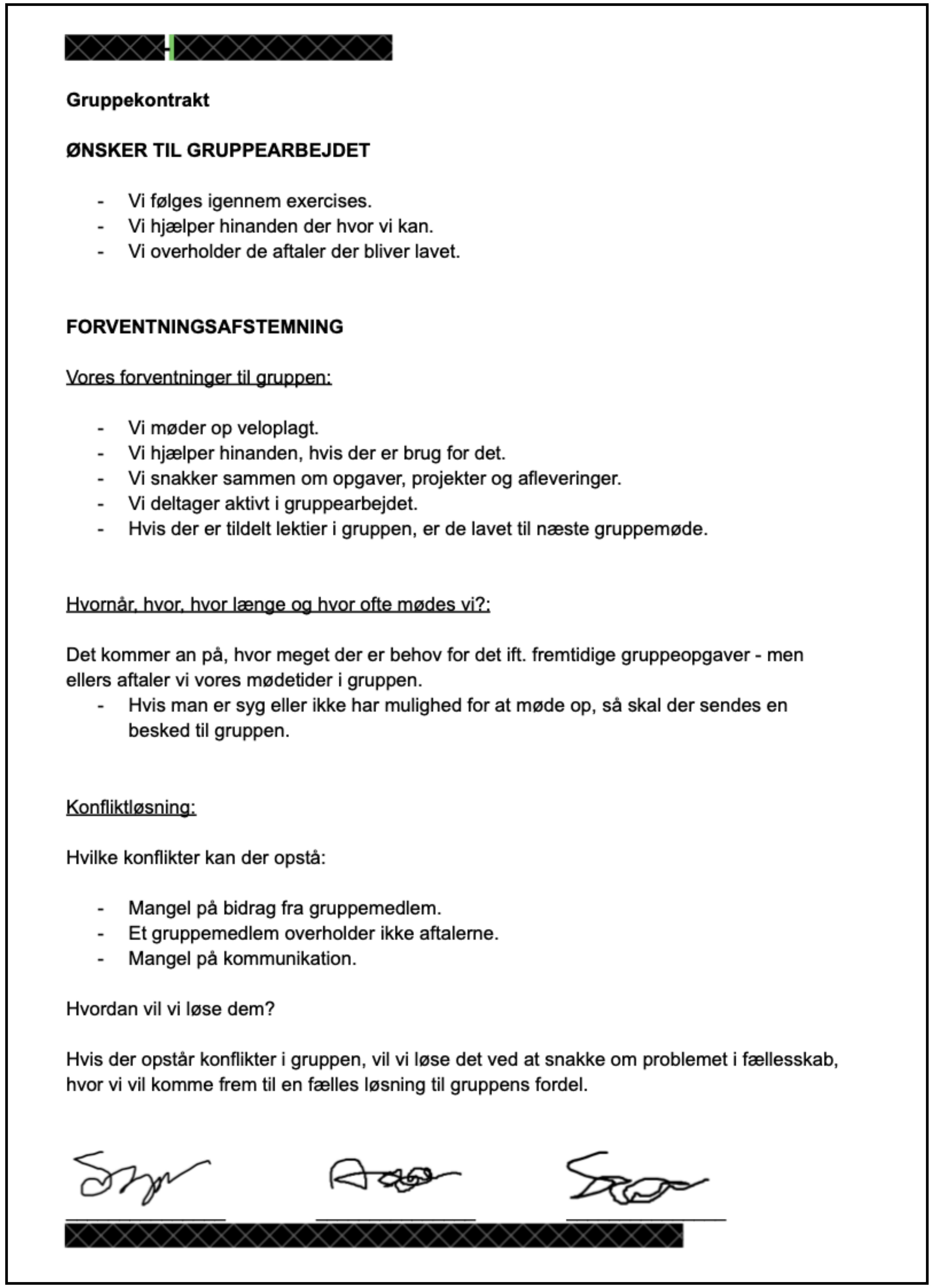
Applikationsudvikling: CS101

Control flow & Lambda expressions

NIFR 2024 - https://github.com/nicklasdean/ita-2023-2sem-code-examples

Gruppekontrakt
Fronter



[redacted]

Gruppekontrakt

ØNSKER TIL GRUPPEARBEJDET

- Vi følges igennem exercises.
- Vi hjælper hinanden der hvor vi kan.
- Vi overholder de aftaler der bliver lavet.

FORVENTNINGSAFSTEMNING

Vores forventninger til gruppen:

- Vi møder op veloplagt.
- Vi hjælper hinanden, hvis der er brug for det.
- Vi snakker sammen om opgaver, projekter og afleveringer.
- Vi deltager aktivt i gruppearbejdet.
- Hvis der er tildelt lektier i gruppen, er de lavet til næste gruppemøde.

Hvornår, hvor, hvor længe og hvor ofte mødes vi?:

Det kommer an på, hvor meget der er behov for det ift. fremtidige gruppeopgaver - men ellers aftaler vi vores mødetider i gruppen.

- Hvis man er syg eller ikke har mulighed for at møde op, så skal der sendes en besked til gruppen.

Konfliktløsning:

Hvilke konflikter kan der opstå:

- Mangel på bidrag fra gruppemedlem.
- Et gruppemedlem overholder ikke aftalerne.
- Mangel på kommunikation.

Hvordan vil vi løse dem?

Hvis der opstår konflikter i gruppen, vil vi løse det ved at snakke om problemet i fællesskab, hvor vi vil komme frem til en fælles løsning til gruppens fordel.

[illegible] [illegible] [illegible]

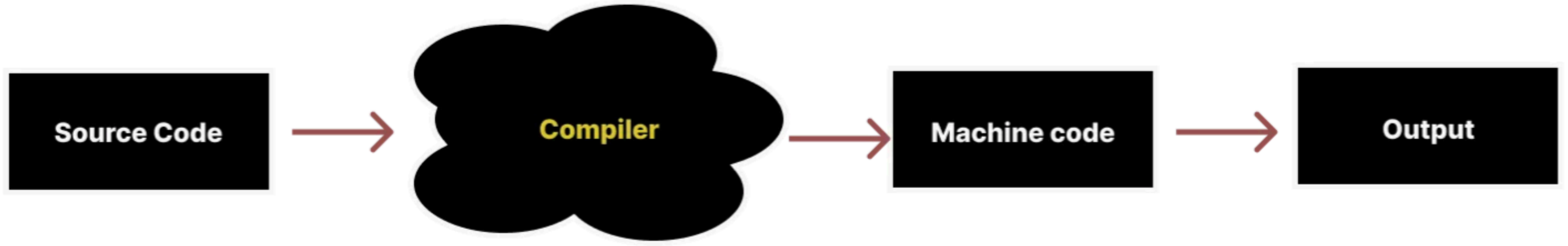
[redacted]

Applikationsudvikling: CS101

Agenda

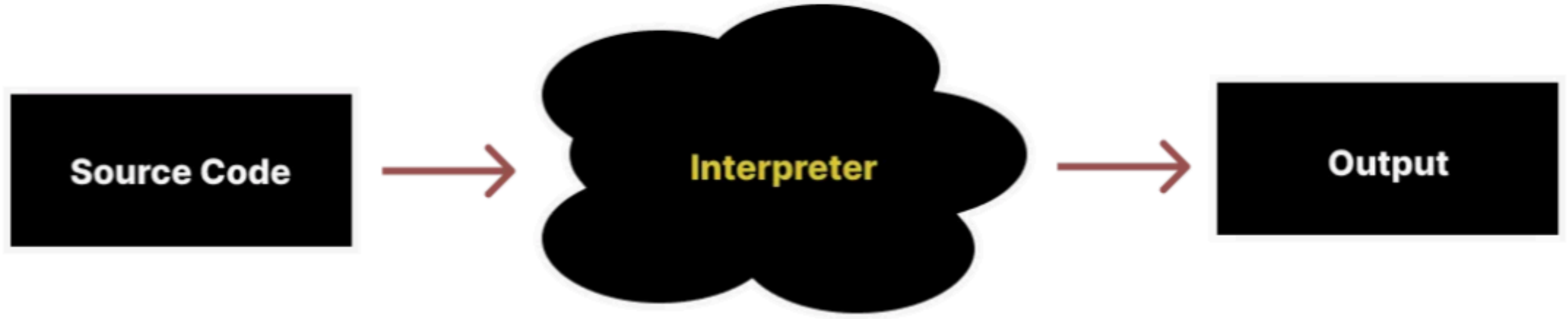
- Compilation/interpretation: Compile-time & Runtime
- Control flow: Boolean logic
- Kotlin
    - Fetching data from the command-line
    - If/Else/When
    - Loops
    - Lambda Expressions
        - Filter/Map

Classic compilation (Kotlin, Java, C++, C#)



https://medium.com/@aamchora/what-exactly-just-in-time-jit-compilation-is-in-javascript-f7aea482843f

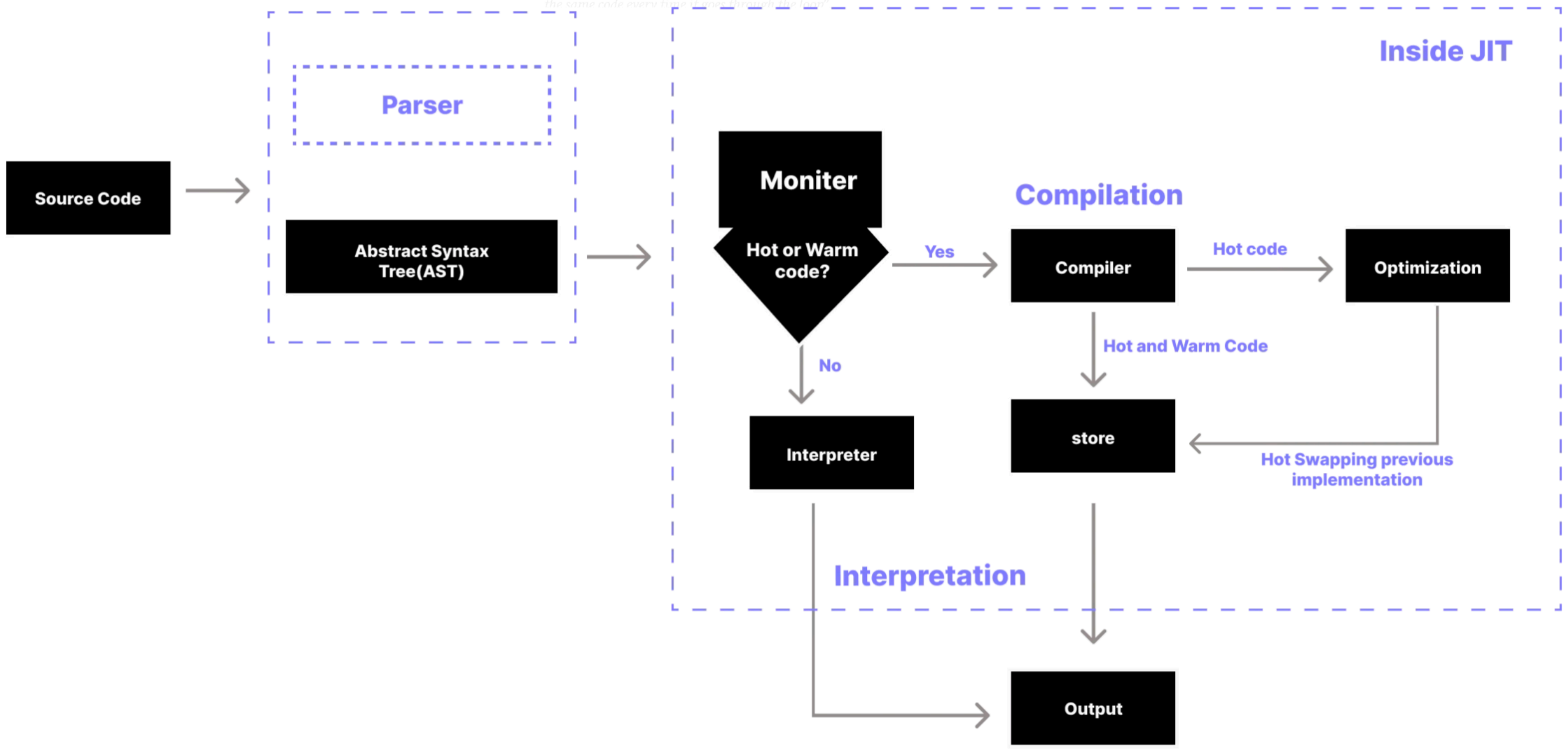
Interpretation (Python, PHP, Perl)



“the interpreter keeps retranslating the same code every time it goes through the loop”

https://medium.com/@aamchora/what-exactly-just-in-time-jit-compilation-is-in-javascript-f7aea482843f

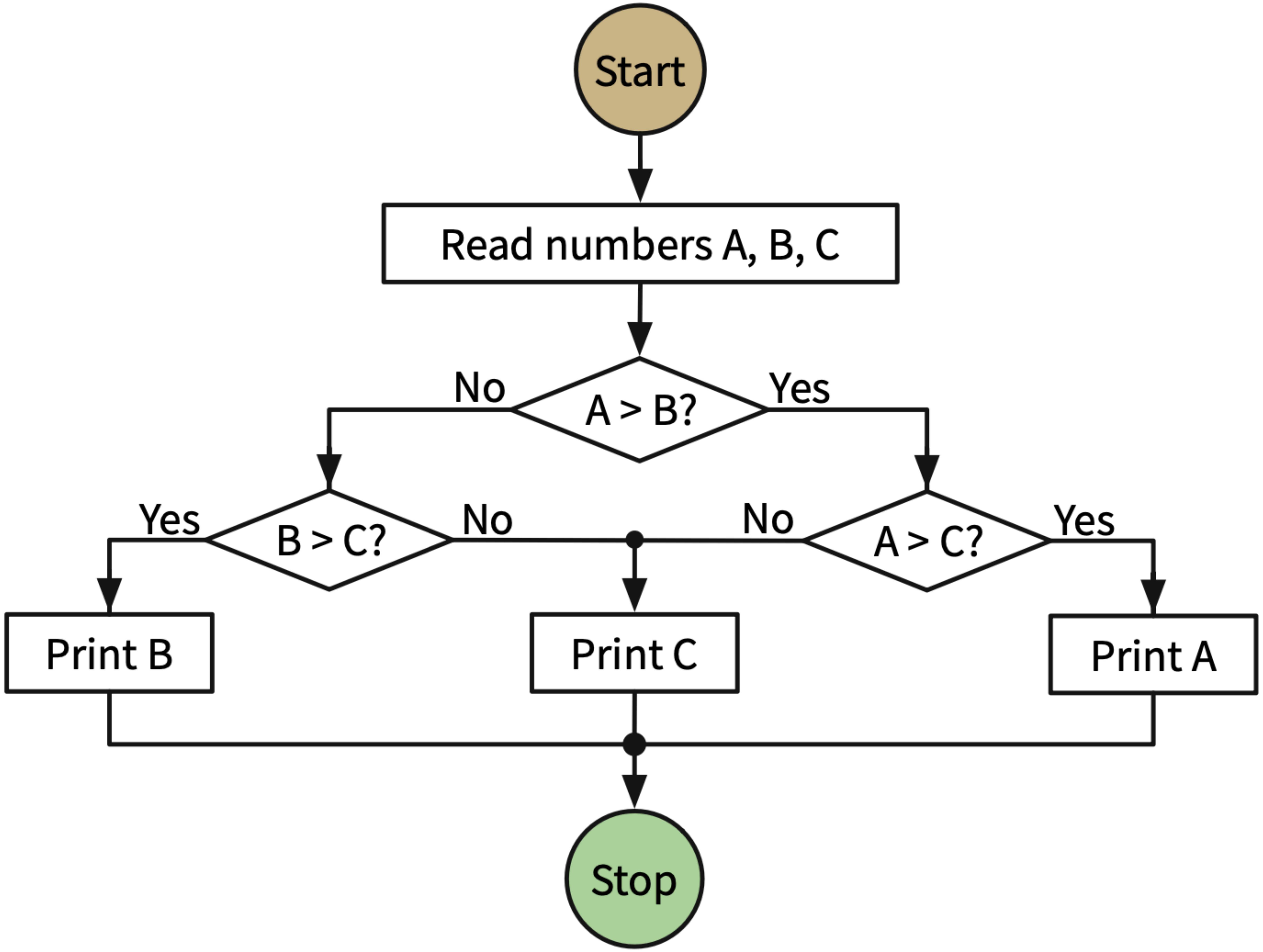
JIT (Javascript)



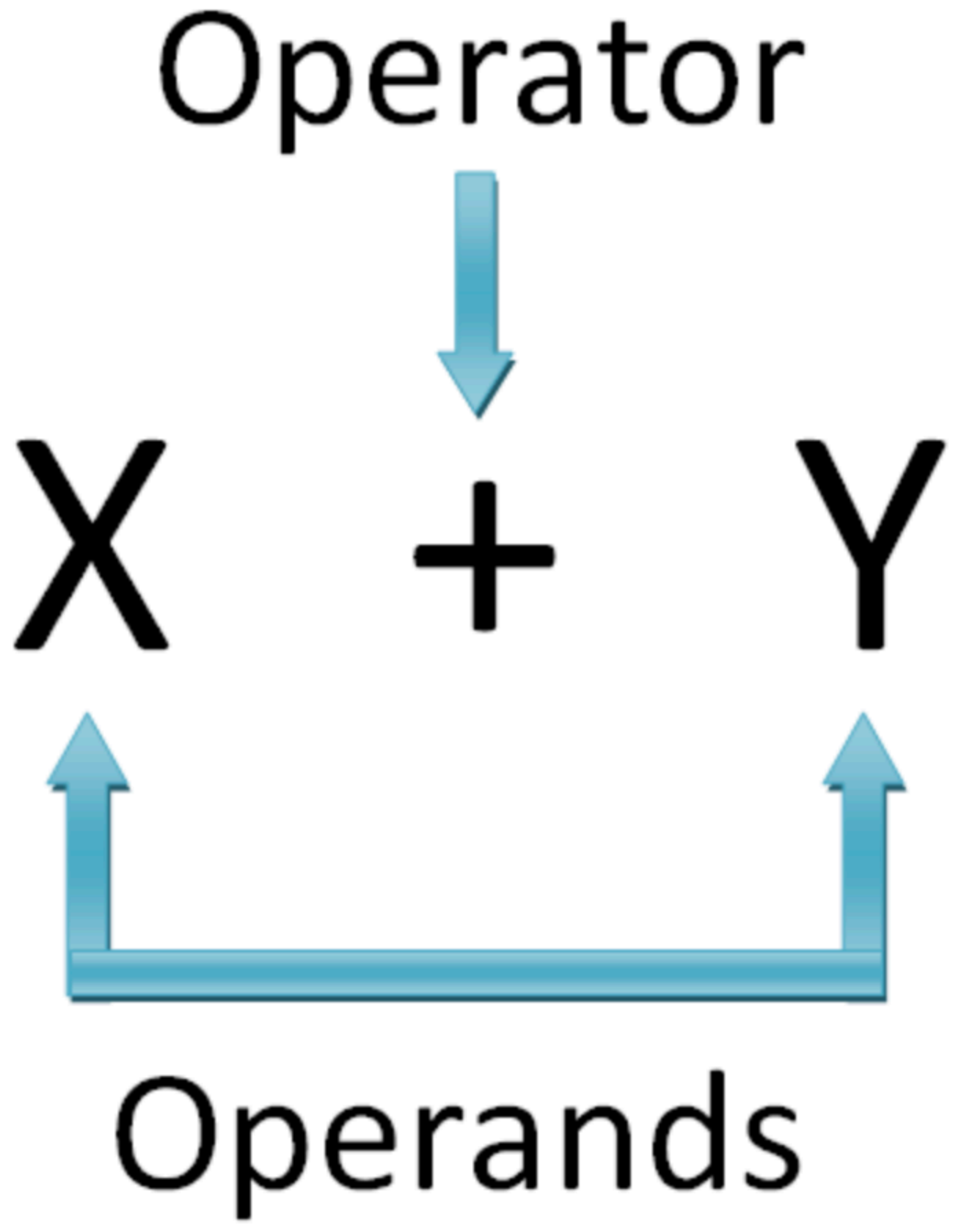
https://medium.com/@aamchora/what-exactly-just-in-time-jit-compilation-is-in-javascript-f7aea482843f

Compilation Example

```
function maximum(A, B, C)
    if A > B
        if A > C
            max ← A
        else
            max ← C
    else
        if B > C
            max ← B
        else
            max ← C
    print max
```



An expression



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Boolean operators

In a nutshell

- Y > X : Y is larger than X
- Y < X : X is larger than Y
- Y =< X : X is larger OR EQUAL to Y
- Y == X : Y is equal to X
- !(Y > X) : Y is NOT larger than X (negation)

Boolean logic

A : The pool is warm.

B : I swim.

Ferreira Filho, W. (2017). *Computer science distilled* (R. Pictet, Ed.). Code Energy.

$A \rightarrow B:$ If the pool is warm, then I’ll swim.

Ferreira Filho, W. (2017). *Computer science distilled* (R. Pictet, Ed.). Code Energy.

Chaining operations

Person skal være over 18 og have medlemskort eller betale indgang

Associativity Parentheses are irrelevant for sequences of **AND** or **OR** operations. As sequences of sums or multiplications in elementary algebra, they can be calculated in any order.

$$A \text{ AND } (B \text{ AND } C) = (A \text{ AND } B) \text{ AND } C.$$
$$A \text{ OR } (B \text{ OR } C) = (A \text{ OR } B) \text{ OR } C.$$

Ferreira Filho, W. (2017). *Computer science distilled* (R. Pictet, Ed.). Code Energy.

DISTRIBUTIVITY In elementary algebra we factor multiplicative terms from sums: $a \times (b + c) = (a \times b) + (a \times c)$. Likewise in logic, **AND**ing after an **OR** is equivalent to **OR**ing results of **AND**s, and vice versa:

$$A \textbf{ AND } (B \textbf{ OR } C) = (A \textbf{ AND } B) \textbf{ OR } (A \textbf{ AND } C).$$
$$A \textbf{ OR } (B \textbf{ AND } C) = (A \textbf{ OR } B) \textbf{ AND } (A \textbf{ OR } C).$$

Ferreira Filho, W. (2017). *Computer science distilled* (R. Pictet, Ed.). Code Energy.

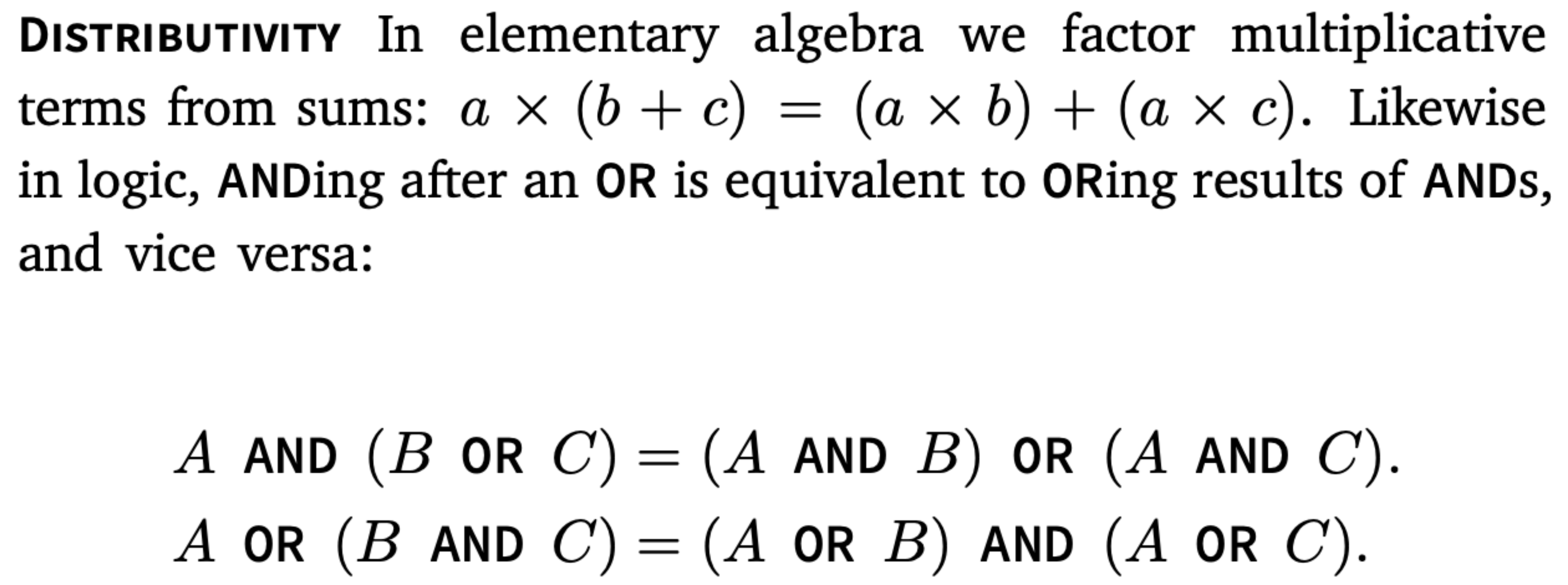
Person skal være over 18 og have medlemskort eller betale indgang

Person skal være over 18 AND have medlemskort OR betale indgang

Person skal være over 18 AND have medlemskort OR betale indgang

2 Predicates

Person skal være over 18 AND (have medlemskort OR betale indgang)



Distributivity In elementary algebra we factor multiplicative terms from sums: $a \times (b + c) = (a \times b) + (a \times c)$. Likewise in logic, **AND**ing after an **OR** is equivalent to **OR**ing results of **AND**s, and vice versa:

$$A \text{ AND } (B \text{ OR } C) = (A \text{ AND } B) \text{ OR } (A \text{ AND } C).$$
$$A \text{ OR } (B \text{ AND } C) = (A \text{ OR } B) \text{ AND } (A \text{ OR } C).$$

Et hit skal have et fængende omkvæd og en god hook og god produktion

Et hit skal have et fængende omkvæd og en god hook og god produktion

Associativity Parentheses are irrelevant for sequences of **AND** or **OR** operations. As sequences of sums or multiplications in elementary algebra, they can be calculated in any order.

$$A \textsf{ AND } (B \textsf{ AND } C) = (A \textsf{ AND } B) \textsf{ AND } C.$$
$$A \textsf{ OR } (B \textsf{ OR } C) = (A \textsf{ OR } B) \textsf{ OR } C.$$

If statements

```
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

Function returning boolean value

```
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

Function signature

```
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

If statement

```
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

Predicate / Expression

$$A \rightarrow B: \text{ If the pool is warm, then I'll swim.}$$

```
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

If block w. scope

Example: isAdult & isMember

Boolean expressions as return values

Control flow

- Full boolean expressions can be used as return values
- A brief way to return values from a function can be to return the value of an expression instead of an explicit value
- Common practice pros and cons: Readability, debug and concise vs verbose

```
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

```
fun isAdult(age: Int): Boolean {
    return age > 18;
}
```

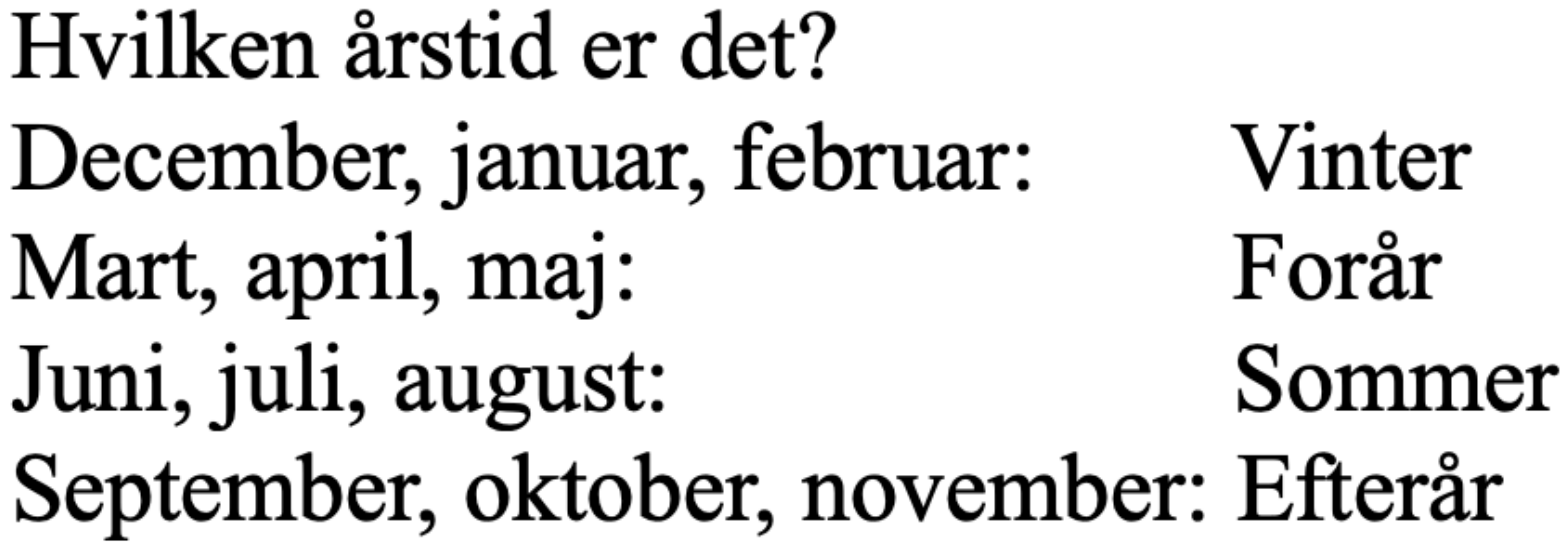
```kotlin
fun isAdult(age: Int): Boolean {
    val isAdult: Boolean;
    if (age > 18) {
        isAdult = true;
    } else {
        isAdult = false;
    }
    return isAdult;
}
```

```kotlin
fun isAdult(age: Int): Boolean {
    return age > 18;
}
```

When statements

When statement

Also known as “switch” statements



Hvilken årstid er det?

December, januar, februar:	Vinter
Mart, april, maj:	Forår
Juni, juli, august:	Sommer
September, oktober, november:	Efterår

```kotlin
val color: String = getColor();

when (color) {
    "red" -> {
        println("User inputted red");
    }
    "blue" -> {
        println("User inputted blue")
    }
    else -> {
        println("User a color that is not blue or red")
    }
}
```

```kotlin
fun Request.getBody() =
    when (val response = executeRequest()) {
        is Success -> response.body
        is HttpError -> throw HttpException(response.status)
    }
```

https://kotlinlang.org/docs/control-flow.html#when-expression

LoopsLoopsLoopsLoopsLoopsLoo
psLoopsLoopsLoopsLoopsLoopsLo
opsLoopsLoopsLoopsLoopsLoops

Loops

Numeric iteration

```
for (i in 1..3) {
    println(i)
}
```

Loops

Iteration in collection

```
for (item: Int in ints)
    // ...
}
```

Example

Exercises 02: Control Flow

Higher order functions & Lambda functions

Functions in Kotlin

Higher order functions

- In Kotlin functions are first class citizens (like javascript)
- This means the language supports passing functions as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures
- A higher order function is a function that takes another function as argument

```kotlin
fun higherOrderFunction(func: () -> Unit) {
    println("Before calling the passed function")
    func()
    println("After calling the passed function")
}

fun sayHello() {
    println("Hello, world!")
}

higherOrderFunction(::sayHello)
```

Why?

Asynchronous programming

```javascript
const url = 'https://jsonplaceholder.typicode.com/posts/1';

fetch(url)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Data:', data);
  })
  .catch(error => {
    console.error('There was a problem with the fetch operation:', error);
  });
```

Arrow function (callback)

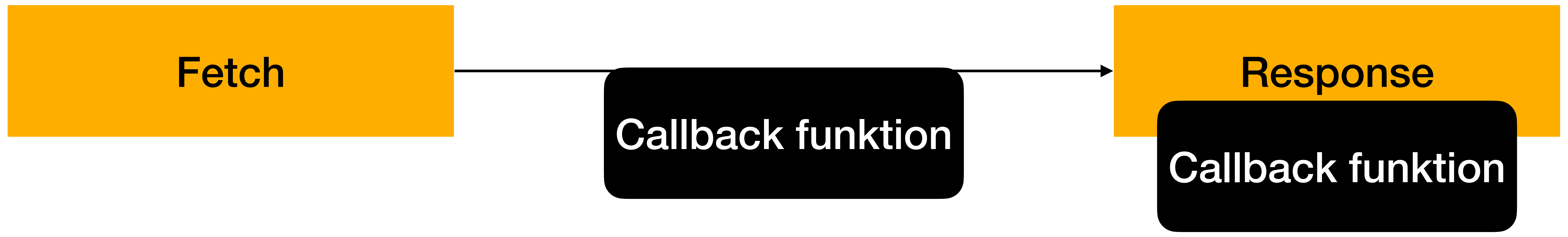
Asynchronous programming

```
const url = 'https://jsonplaceholder.typicode.com/posts/1';

fetch(url)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Data:', data);
  })
  .catch(error => {
    console.error('There was a problem with the fetch operation:', error);
  });
```

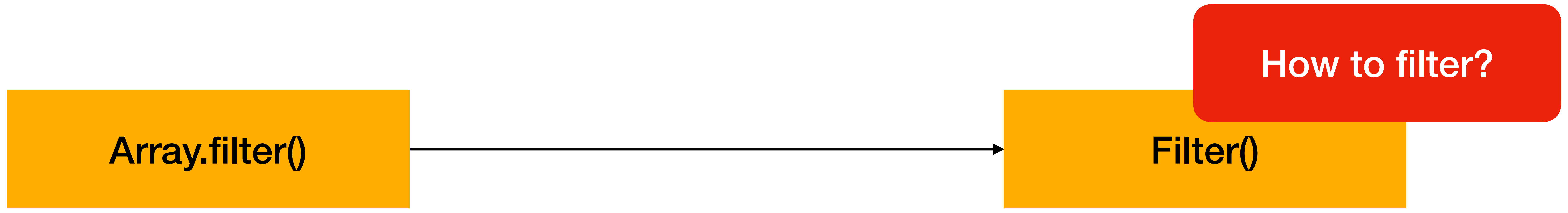
Why?

Asynchronous programming



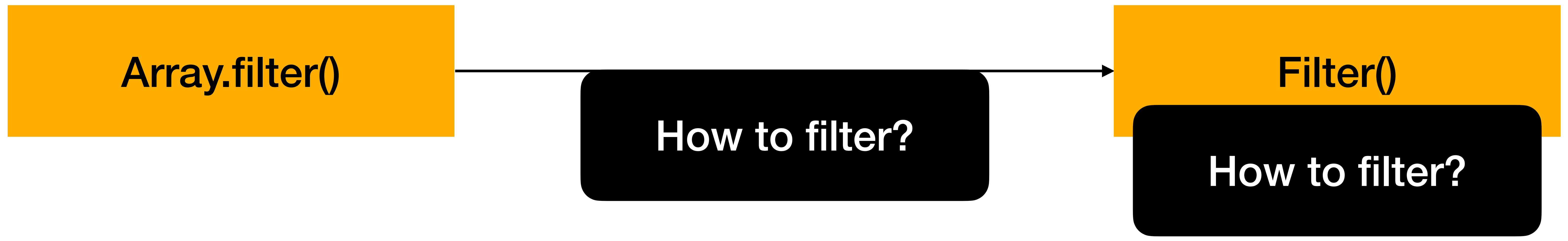
Why?

Flexibility



Why?

Flexibility



Filter: Remove numbers above 5

Predicate

1, 2, 3, 4, 5, 6, 7, 8, 9

🟥 🟥 🟥 🟥 🟥 🟩 🟩 🟩 🟩

Lambda functions example