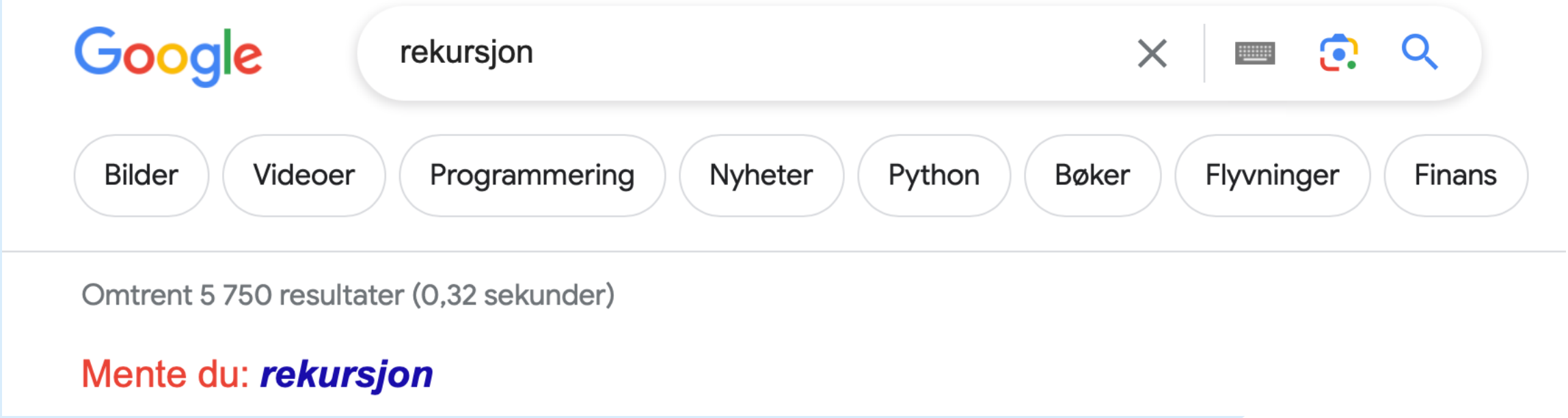



# Rekursjon

Uke 7 IN1010

Agneshb@uio.no

# I dag

- Rekursive metoder
- Call stack i rekursjon – med tegning på hvis dere vil?

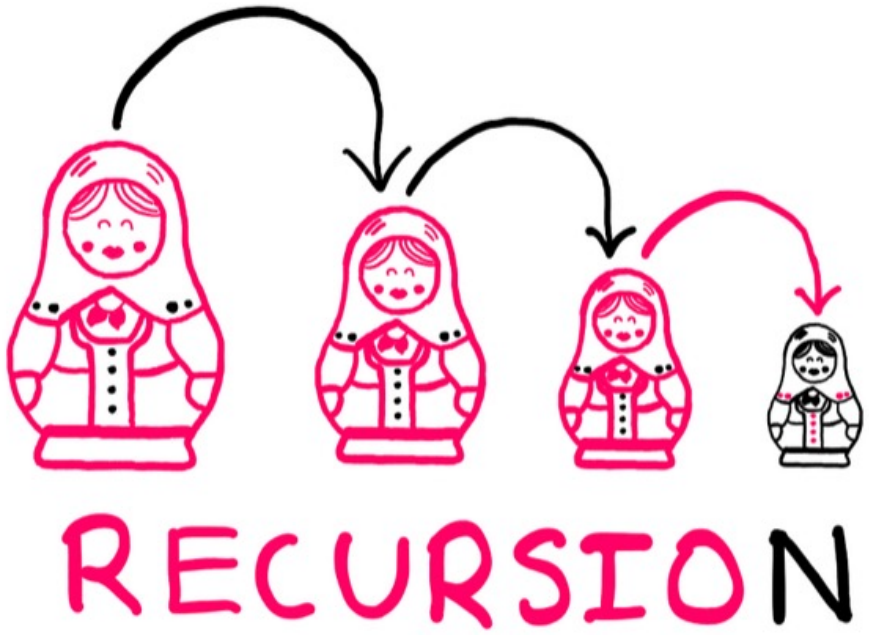
- Livekoding av en metode som teller antall hunder eller katter i en lenkeliste av Dyr.
- Livekoding av fibonacci implementert rekursivt

- Oppgaver



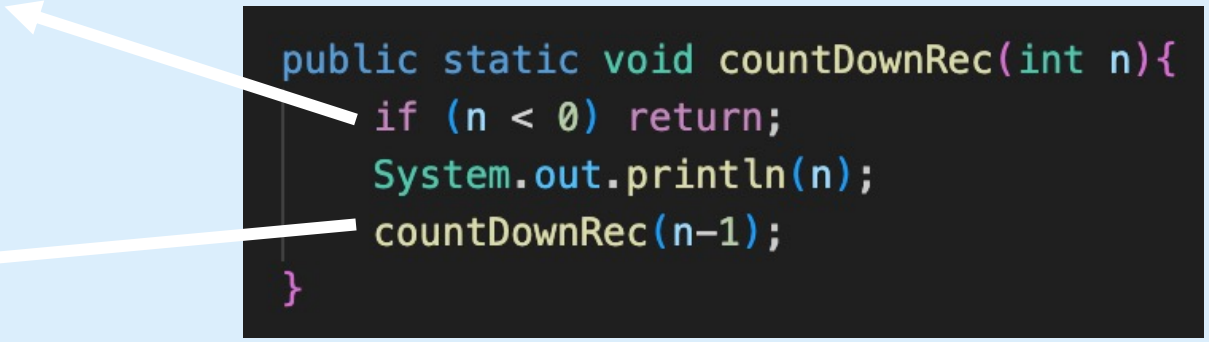
# Rekursive metoder

- Enkelt forklart: rekursive metoder er metoder som kaller på seg selv.
  - Som igjen kaller seg selv
    - Som igjen kaller seg selv
      - (..)
- Kan fortsette evig, som en evig while-løkke
- Ønsker å stoppe en gang
- Iterative løsninger kan omformuleres til rekursive løsninger.



# Hva inneholder en rekursiv metode?

- **Basistilfelle** (“base case”)
  - Rekursjonen stopper, kaller ikke seg selv igjen → «stop case».
  - Kallene «nøstes opp» fra siste til første kall.
- **Rekursjonssteg** (“recursive step”)
  - Endrer noe slik at man til slutt når basistilfellet.
  - I dette tilfelle er det ett rekursivt kall, men en det kan være flere recursive kall (feks Fibonacci).



```
public static void countDownRec(int n){
    if (n < 0) return;
    System.out.println(n);
    countDownRec(n-1);
}
```

# Rekursiv vs. iterativ

**Rekursiv**

```java
public static void countDownRec(int n){
    if (n < 0) return;
    System.out.println(n);
    countDownRec(n-1);
}
```

**Iterativ**

```java
public static void countDownIt(int n) {
    while(n >= 0) {
        System.out.println(n);
        n--;
    }
}
```

countDownRec(5)

countDownIt(5)



```
5
4
3
2
1
0
```

# Hva skjer i den rekursive metoden?

```java
public static void countDownRec(int n){
    if (n < 0) return;
    System.out.println(n);
    countDownRec(n-1);
}
```

```
countDownRec(n:5);
```

**n = 5** → printer **5**
→ kaller countDownRec(**4**)

**n = 4** → printer **4**
→ kaller countDownRec(**3**)

**n = 3** → printer **3**
→ kaller countDownRec(**2**)

**n = 2** → printer **2**
→ kaller countDownRec(**1**)

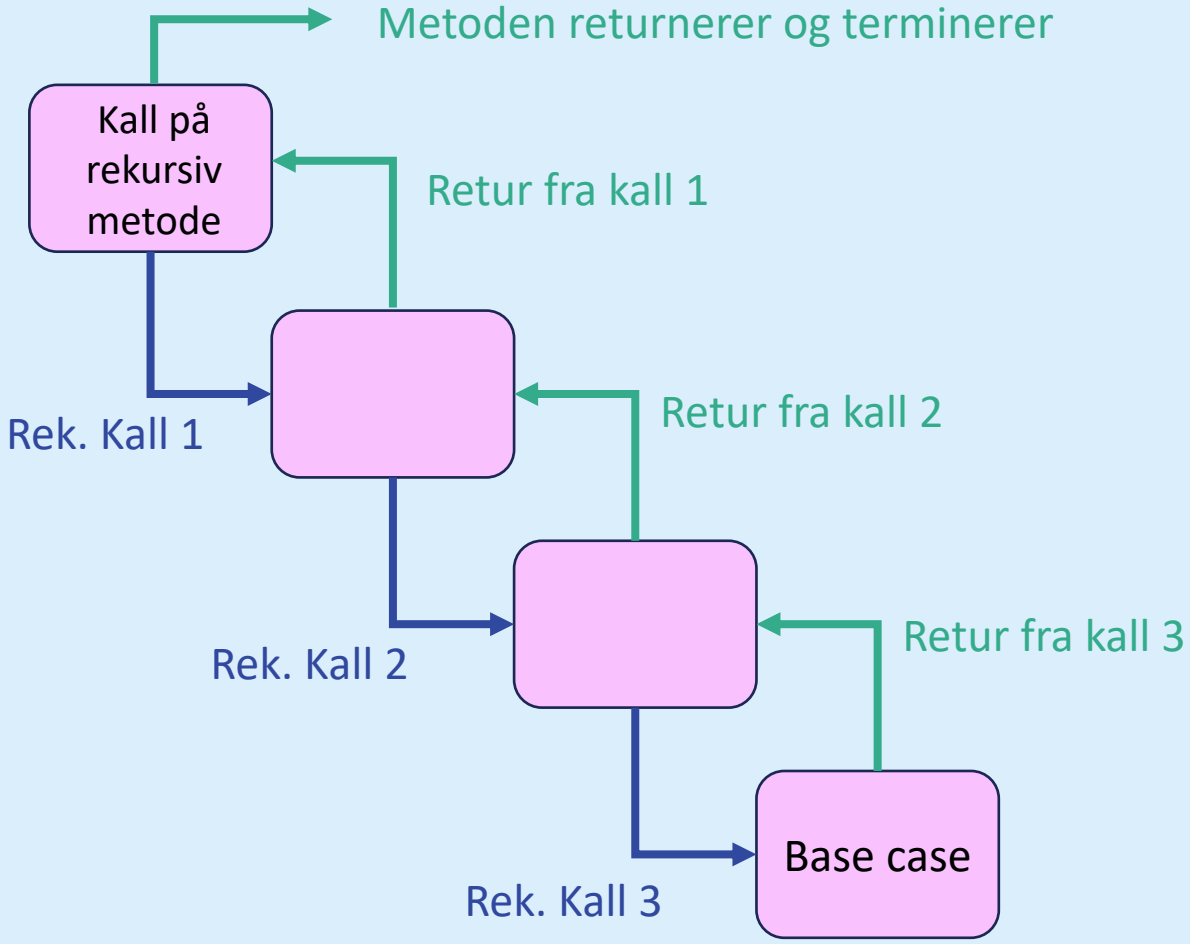
**n = 1** → printer **1**
→ kaller countDownRec(**0**)

**n = 0** → printer **0**
→ kaller countDownRec(**-1**)

**n = -1** → n er mindre enn 0 og returnerer uten rekursivt kall

# Call stack i forbindelse med rekursjon

- Stack (Stabel):
  - Push() – legger til foran
  - Pop() – henter ut foran
- Rekursive metodekall legger seg på en **call stack**.
- Når vi når basistilfellet stopper rekursjonen, og alle elementene poppes fra stacken.
  - Det siste metodekallet poppes først og det første metodekallet poppes sist
- Return: kan enten returnere en verdi eller terminere metodekallet.



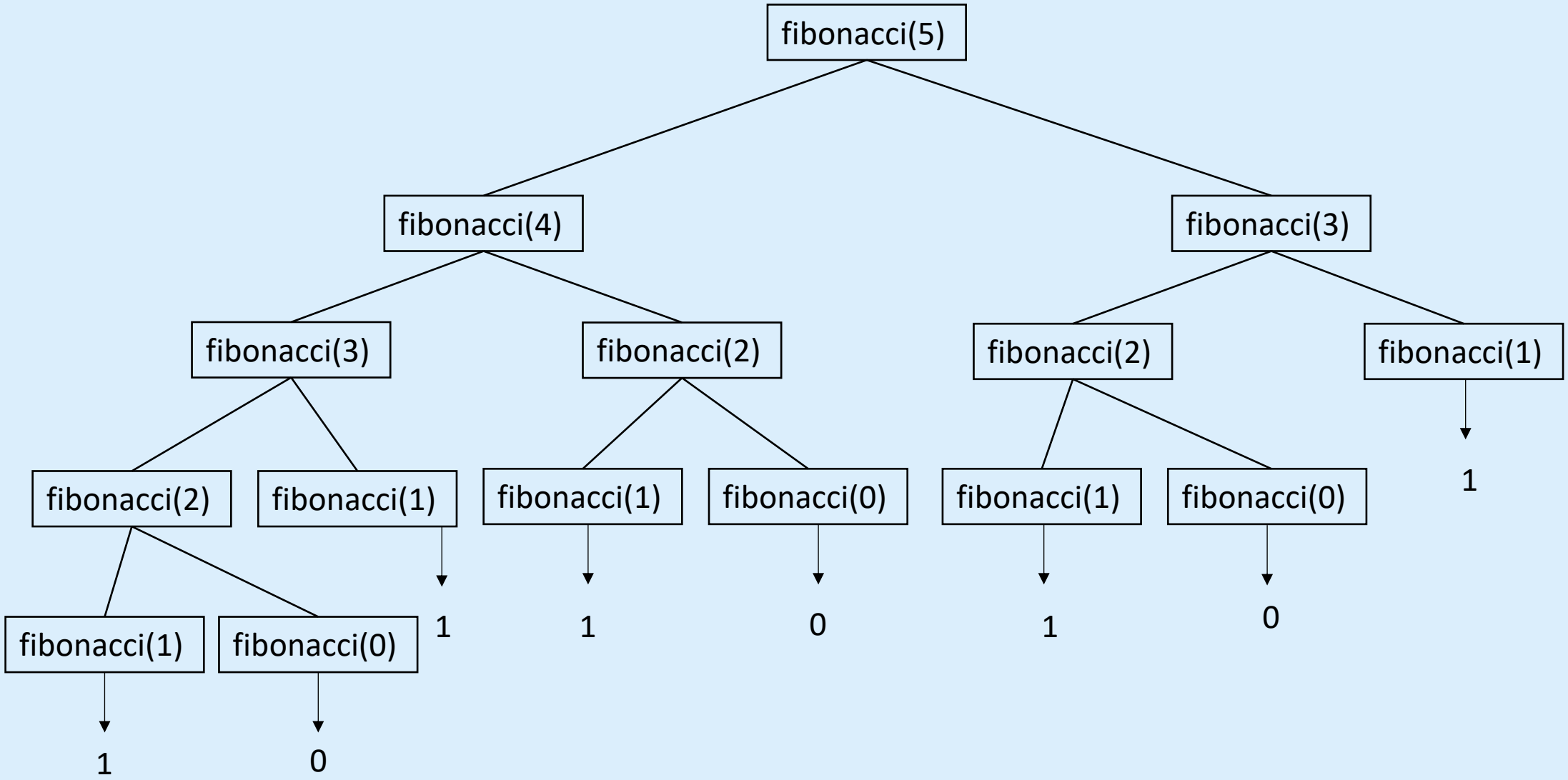
# Eksempel: rekursiv metode for å summere tall fra 0 til og med n

```java
public static int recursiveSum(int n) {
    // Base case: når n = 0 er det ikke flere tall å summere
    if (n == 0) {
        return 0;
    }
    // Recursive step: summerer n med resultatet av et
    //rekursivt kall på recursiveSum med n-1 som argument
    return n + recursiveSum(n-1);
}
```

recursiveSum(5)
→ 5 + recursiveSum(4)
→ 5 + 4 + recursiveSum(3)
→ 5 + 4 + 3 + recursiveSum(2)
→ 5 + 4 + 3 + 2 + recursiveSum(1)
→ 5 + 4 + 3 + 2 + 1 + recursiveSum(0)
→ 5 + 4 + 3 + 2 + 1 + 0
→ 5 + 4 + 3 + 2 + 1
→ 5 + 4 + 3 + 3
→ 5 + 4 + 6
→ 5 + 10
→ 15

# Livekoding

Fibonacci


fibonacci(5)
fibonacci(4)
fibonacci(3)
fibonacci(3)
fibonacci(2)
fibonacci(2)
fibonacci(1)
fibonacci(2)
fibonacci(1)
fibonacci(1)
fibonacci(0)
fibonacci(1)
fibonacci(0)
1
fibonacci(1)
fibonacci(0)
1
1
0
1
0
1
1
0

# Til slutt

- Dette må du vite:
  - Rekursive metoder må ha med base case og rekursivt steg
- Rekursjon er vanskelig i begynnelsen, men det er lurt å lære seg til eksamen (ref tidligere eksamensoppgaver)
  - Løs ukesoppgaver, Trix og innlevering 3
- Positivt: når man først har funnet base case og rekursivt steg, kreves det ofte få linjer kode☺

# Jobbe med oppgaver<3

- Seminaroppgaver finnes på gruppesiden til gruppe 3
- Trix
- Innlevering 2 (men gjør gjerne noen mindre oppgaver først!)
- Prøv å tegn rekursive kall, eller gjette hva rekursive kall returnere/printer ut (øvelse til midtveis)