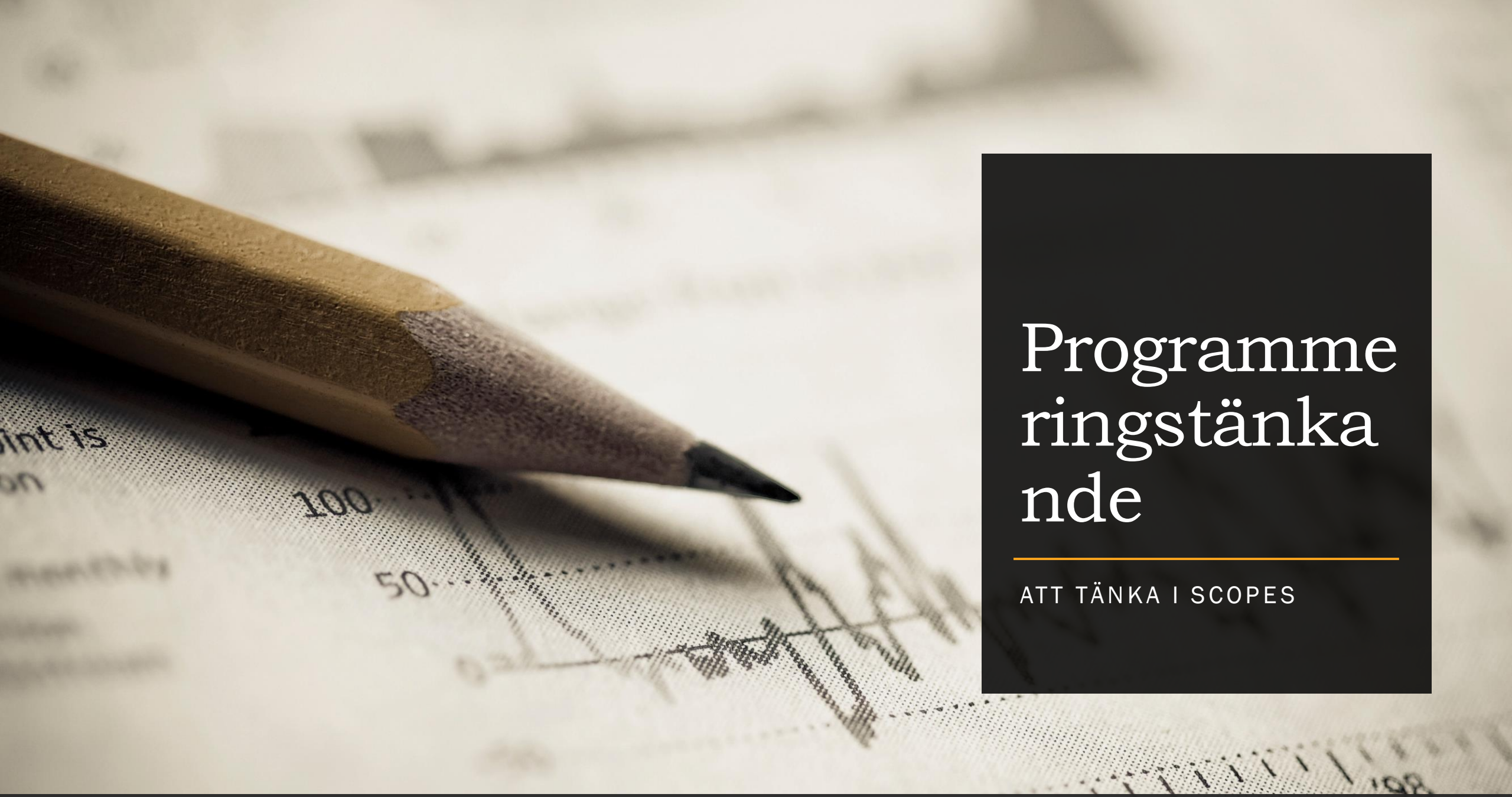

Programmeringstänkande

ATT TÄNKA I SCOPES

Vad är ett Scope

Scope betyder omfattning, räckvidd.

I programmering är ett scope en avdelning i koden, det kan även betyda en avgränsad del av projektet med sina egenskaper och funktioner.

En modul-fil är ett scope för sig

Måsvingar markerar ett scope

Som regel så dör allt innanför ett scope när scopen avslutas.

Scope levnadstid

Så länge en modul eller class hålls vid liv så är scopen vid liv.

Den stunden man avslutar arbetet med det, eller sätter variabeln till null, så tappar vi kopplingen till det scopet och då kommer allt som lagrats i minnet av scopen att raderas.

Varför sker detta?

Kodexempel

```
function SayHello()
{
    let greeting="Hello";
    console.log(greeting);
}
```

Variabeln Greeting lever bara så länge som scopet finns. Efter att scopet avslutats kan man inte nå variabeln längre.

*(detta gäller inte för **var**, därför använder vi inte var)*

Liv efter Scope

```
function GenerateHelloToUser(username:string)
{
    let greeting="Hello " + username;
    return greeting;
}

let hello=GenerateHelloToUser("Lars");
```

I detta fall kommer greeting att leva vidare för den skickas ut innan funktionen och Scopet avslutas. Greeting kommer att leva vidare i variabeln Hello.

Då greeting är en sträng kommer den att kopieras till hello iofs… men den lever i alla fall vidare.

Variabler

```
let name = { name: "Lars" }
let clone = name
clone.name = "Mr Lars"
name.name += "!"
console.log(name)
```

Variabler lagras i minnet.

Objektvariabeln name innehåller inte ”name:Lars” egentligen, den innehåller en address till en plats i minnet där det står *name: Lars*. Den innehåller en referens.

När vi nu kopierar den kommer adressen till platsen i minnet att kopieras från name till clone. Så om någon av variablerna ändras, kommer båda att påverkas.

Minnet är kort

Man kopierar referenser till objekt för att slippa skapa om objekt, det kostar tid och minnesplats.

Av samma orsak vill man inte att objekt ska leva vidare efter en avslutad scope. Det kostar minne och efter några körningar kan minnet ta slut, eller så blir datorn långsam.

Tänk om man skulle spara alla rester i kylskåpet utan att någonsin tänka på att de finns där. Med tiden blir kylskåpet fullt med rutten mat. Alltså måste man rensa i kylskåpet och minnet.

Minnet utanför scopen

Det finns en speciell regel när det gäller scope.

Man kan nå alla variabler i scopet ovanför, men inte nedan.

```
var greeting="Hello“
for(let count=0; count<3; count++)
{
  greeting+="!";
}
console.log(greeting);
```

Variabeln greeting finns i en scope ovanför det som finns i for-satsen, men for-satsen kan nå variabeln ändå. Om vi nu hade frestats att skriva i slutet

```
console.log(count);
```

så skulle programmet kraschat. För variabeln count lever inte längre efter att for-loopen avslutats. Detta gäller dock inte om man använt **var** istället för **let** i for-loopen.

Kodmönster

Man följer alltid samma mönster inom programmering

1. Deklarera variabler
2. Kontrollera eller sätt variablernas värde
3. Använd variablerna
4. Presentera / returnera resultat

Exempel

Den här enkla loopen är ett bra exempel.

```
for (let count = 0; count < 10; count++) {
    console.log(count);
}
```

Först skapar man variabeln count. Sedan ger man den ett värde. Efter det kontrollerar man (vid varje loop runda) att den är inom acceptabla gränser. Slutligen bearbetar vi den genom att öka med ett och presenterar resultatet.

Ett annat exempel

```
let age=25;
let bmonth=8;
let bday=10;
let today = new Date().getDate();
let thisMonth = new Date().getMonth();
if (bmonth>=thisMonth && bday>=today)
{
    age++;
}
console.log(age);
```

Ett exempel till

```
const testingNonExistingRoute = () => {
    describe('Test a route that does not exist', () => {
        test('Expecting 404 not found', (done:any) => {
            Chai.request(app)
                .get(`/${ randomString }`)
                .end((req, res) => {
                    res.should.have.a.status(404)
                    done()
                })
        })
    })
}
```