Informasjonsteknologi 1 og 2 Hjelpeark

VG2 (IT1) og VG3 (IT2) — LK20

IT1: Nettverk og internett

OSI-modellen (7 lag)

1. **Fysisk** – kabler, signaler
2. **Datalink** – MAC-adresser
3. **Nettverk** – IP-adresser, routing
4. **Transport** – TCP/UDP, porter
5. **Sesjon** – forbindelser
6. **Presentasjon** – kryptering
7. **Applikasjon** – HTTP, SMTP

TCP/IP-modellen (4 lag)

- **Nettverksgrensesnitt** – fysisk
- **Internett** – IP
- **Transport** – TCP/UDP
- **Applikasjon** – HTTP, DNS

IP-adresser

- **IPv4:** 32 bit, f.eks. 192.168.1.1
- **IPv6:** 128 bit, f.eks. 2001:db8::1
- **Private:** 10.x.x.x, 172.16-31.x.x, 192.168.x.x
- **Loopback:** 127.0.0.1 (localhost)

DNS (Domain Name System)

- Oversetter domenenavn til IP
- Hierarkisk: root → TLD → domene
- TLD: .no, .com, .org, .net
- A-record: domene → IPv4

Protokoller

- **HTTP/HTTPS** – web (port 80/443)
- **FTP** – filoverføring (port 21)
- **SMTP** – e-post sending (port 25)
- **SSH** – sikker tilgang (port 22)
- **TCP** – pålitelig, rekkefølge
- **UDP** – rask, ingen garanti

IT1: HTML og CSS

HTML-struktur

```
<!DOCTYPE html>
<html lang="no">
<head>
  <meta charset="UTF-8">
  <title>Tittel</title>
  <link rel="stylesheet" href="stil.css">
</head>
<body>
  <header>Topptekst</header>
  <main>Hovedinnhold</main>
  <footer>Bunntekst</footer>
</body>
</html>
```

Semantiske elementer

- `<header>` – topptekst
- `<nav>` – navigasjon
- `<main>` – hovedinnhold
- `<article>` – selvstendig innhold
- `<section>` – tematisk del
- `<aside>` – sidepanel
- `<footer>` – bunntekst

CSS-selektorer

```
p { color: blue; }

/* Klasse */
.klasse { margin: 10px; }

/* ID */
#id { padding: 20px; }

/* Kombinert */
nav a { text-decoration: none; }
```

CSS Flexbox

```
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row;
  gap: 10px;
}
```

CSS Grid

```
  display: grid;
  grid-template-columns: 1fr 2fr;
  grid-gap: 20px;
}
```

Responsivt design

```
  .container {
    flex-direction: column;
  }
}
```

IT1: Programmering

Variabler og datatyper

```
navn = "Ola"        # string
alder = 17          # int
hoyde = 1.75        # float
aktiv = True        # boolean
liste = [1, 2, 3]   # list
```

```
let navn = "Ola";
const PI = 3.14;
let tall = [1, 2, 3];
```

Kontrollstrukturer

```
if alder >= 18:
    print("Voksen")
elif alder >= 13:
    print("Tenåring")
else:
    print("Barn")

# For-løkke
for i in range(5):
    print(i)

# While-løkke
while betingelse:
    # gjør noe
```

Funksjoner

```
    return f"Hei, {navn}!"

resultat = hils("Kari")
```

```
  return `Hei, ${navn}!`;
}

const pil = (x) => x * 2;
```

Lister og arrays

```
tall.append(6)    # legg til
tall.pop()        # fjern siste
len(tall)         # lengde
tall[0]           # første element
tall[-1]          # siste element
```

IT1: Databaser og SQL

Grunnbegreper

- **Tabell** – rader og kolonner
- **Rad** – en post/entitet
- **Kolonne** – et attributt
- **Primærnøkkel** – unik ID
- **Fremmednøkkel** – relasjon

SQL-spørringer

```
SELECT * FROM elever;
SELECT navn, alder FROM elever
WHERE alder > 16
ORDER BY navn;

-- Sette inn
INSERT INTO elever (navn, alder)
VALUES ('Ola', 17);

-- Oppdatere
UPDATE elever
SET alder = 18
WHERE navn = 'Ola';

-- Slette
DELETE FROM elever
WHERE id = 5;
```

JOIN

```
FROM elever e
INNER JOIN klasser k
ON e.klasse_id = k.id;
```

Aggregatfunksjoner

```
SELECT AVG(alder) FROM elever;
SELECT MAX(poeng) FROM resultater;
SELECT klasse, COUNT(*)
FROM elever GROUP BY klasse;
```

IT1: Informasjonssikkerhet

CIA-triaden

- **Konfidensialitet** – kun autorisert tilgang
- **Integritet** – data er korrekte
- **Tilgjengelighet** – systemet fungerer

Trusler

- **Malware:** virus, trojan, ransomware
- **Phishing:** falske e-poster/nettsider
- **DDoS:** overbelastningsangrep
- **SQL-injection:** ondsinnet kode i input
- **Man-in-the-middle:** avlytting

Beskyttelse

- Sterke passord, 2FA
- HTTPS og kryptering
- Brannmur og antivirus
- Oppdateringer og patching
- Backup og sikkerhetskopiering

Kryptering

- **Symmetrisk:** samme nøkkel (AES)
- **Asymmetrisk:** offentlig/privat (RSA)
- **Hashing:** enveisfunksjon (SHA-256)

IT1: Personvern og etikk

GDPR-prinsipper

- Lovlighet, rettferdighet, åpenhet
- Formålsbegrensning
- Dataminimering
- Riktighet
- Lagringsbegrensning
- Integritet og konfidensialitet

Rettigheter

- Innsyn i egne data
- Retting av feil
- Sletting (“retten til å bli glemt”)
- Dataportabilitet
- Protest mot behandling

Etiske problemstillinger

- Overvåking vs. sikkerhet
- Algoritmebasert diskriminering
- Digital ulikhet
- Opphavsrett og deling

IT2: Objektorientert programmering

Klasser og objekter

```
    def __init__(self, navn, alder):
        self.navn = navn
        self.alder = alder

    def hils(self):
        return f"Hei, jeg er {self.navn}"

elev1 = Elev("Ola", 17)
print(elev1.hils())
```

Arv (inheritance)

```
    def __init__(self, navn):
        self.navn = navn

class Student(Person):
    def __init__(self, navn, skole):
        super().__init__(navn)
        self.skole = skole
```

Innkapsling

```
    def __init__(self, saldo):
        self._saldo = saldo  # protected
        self.__pin = 1234    # private

    def hent_saldo(self):
        return self._saldo

    def sett_inn(self, beløp):
        if beløp > 0:
            self._saldo += beløp
```

Polymorfisme

```
    def lyd(self):
        pass

class Hund(Dyr):
    def lyd(self):
        return "Voff!"

class Katt(Dyr):
    def lyd(self):
        return "Mjau!"

dyr = [Hund(), Katt()]
for d in dyr:
    print(d.lyd())
```

IT2: Algoritmer

Søkealgoritmer

```
def lineært_søk(liste, verdi):
    for i, x in enumerate(liste):
        if x == verdi:
            return i
    return -1

# Binært søk - O(log n)
def binært_søk(liste, verdi):
    lav, høy = 0, len(liste) - 1
    while lav <= høy:
        midt = (lav + høy) // 2
        if liste[midt] == verdi:
            return midt
        elif liste[midt] < verdi:
            lav = midt + 1
        else:
            høy = midt - 1
    return -1
```

Sorteringsalgoritmer

```
def boblesortering(liste):
    n = len(liste)
    for i in range(n):
        for j in range(n-i-1):
            if liste[j] > liste[j+1]:
                liste[j], liste[j+1] = \
                    liste[j+1], liste[j]
    return liste

# Innbygget sortering - O(n log n)
sortert = sorted(liste)
liste.sort()
```

Big O-notasjon

- **O(1)** – konstant tid
- **O(log n)** – logaritmisk
- **O(n)** – lineær
- **O(n log n)** – effektiv sortering
- **O(n^2)** – kvadratisk
- **O(2^n)** – eksponentiell

IT2: Datastrukturer

Stakk (Stack) – LIFO

```
stakk.append(1)  # push
stakk.append(2)
topp = stakk.pop()  # 2
```

Kø (Queue) – FIFO

```
kø = deque()
kø.append(1)     # enqueue
kø.append(2)
første = kø.popleft()  # 1
```

Dictionary / HashMap

```
    "e001": {"navn": "Ola", "alder": 17},
    "e002": {"navn": "Kari", "alder": 18}
}
print(elever["e001"]["navn"])
elever["e003"] = {"navn": "Per", "alder": 17}
```

Set (mengde)

```
b = {2, 3, 4}
print(a | b)  # union: {1,2,3,4}
print(a & b)  # snitt: {2,3}
print(a - b)  # differanse: {1}
```

IT2: API-er og webtjenester

REST API

- **GET** – hente data
- **POST** – opprette ny ressurs
- **PUT** – oppdatere ressurs
- **DELETE** – slette ressurs

Hente data fra API

```
# GET-forespørsel
response = requests.get(
    "https://api.example.com/data"
)
data = response.json()

# Med parametere
params = {"q": "oslo", "limit": 10}
response = requests.get(url, params=params)
```

POST-forespørsel

```
response = requests.post(
    "https://api.example.com/elever",
    json=payload
)
```

JSON

```
# Python til JSON
json_str = json.dumps(data)

# JSON til Python
data = json.loads(json_str)
```

Fetch API (JavaScript)

```
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));

// Async/await
async function hentData() {
  const response = await fetch(url);
  const data = await response.json();
  return data;
}
```

IT2: Systemutvikling

Utviklingsmodeller

- **Fossefallsmodellen:** lineær, faseinndelt
- **Smidig (Agile):** iterativ, fleksibel
- **Scrum:** sprinter, daglige møter
- **Kanban:** visualisering, flyt

Utviklingsfaser

1. Kravspesifikasjon
2. Design/planlegging
3. Implementering
4. Testing
5. Utrulling
6. Vedlikehold

Testing

- **Enhetstesting:** teste enkeltfunksjoner
- **Integrasjonstesting:** teste samspill
- **Systemtesting:** teste hele systemet
- **Brukertesting:** teste med brukere

Versjonskontroll (Git)

```
git add .
git commit -m "Melding"
git push origin main
git pull
git branch ny-funksjon
git checkout ny-funksjon
git merge ny-funksjon
```

IT2: Kunstig intelligens

Grunnbegreper

- **AI:** maskiner som simulerer intelligens
- **ML:** læring fra data
- **Deep learning:** nevrale nettverk
- **Trening:** lære fra treningsdata
- **Prediksjon:** forutsi nye data

Typer maskinlæring

- **Veiledet:** merket data (klassifisering, regresjon)
- **Ikke-veiledet:** umerket data (clustering)
- **Forsterkende:** belønning/straff

Enkel ML i Python

```
    LinearRegression

# Tren modell
model = LinearRegression()
model.fit(X_train, y_train)

# Prediker
y_pred = model.predict(X_test)
```

Etikk i AI

- Bias i treningsdata
- Forklarbarhet (“black box”)
- Personvern
- Ansvar ved feil
- Arbeidsplasser og automatisering

Nyttige tips

Debugging

- Les feilmeldinger nøye
- Bruk print() for å sjekke verdier
- Test små deler av gangen
- Bruk debugger i IDE

God kodepraksis

- Beskrivende variabelnavn
- Kommenter kompleks kode

- Del opp i funksjoner
- Følg navnekonvensjoner
- Test koden din

Sjekkliste eksamen

☐ Forstår nettverksprotokoller
☐ Kan skrive HTML/CSS
☐ Behersker grunnleggende programmering
☐ Kan skrive SQL-spørringer
☐ Forstår OOP-konsepter
☐ Kjenner til API-bruk
☐ Forstår sikkerhet og personvern