CT10A0013 Ohjelmointi Pythonilla

L09: Laadunvarmistus, testaaminen, poikkeustenkäsittely

Uolevi Nikula

Päivän asiat

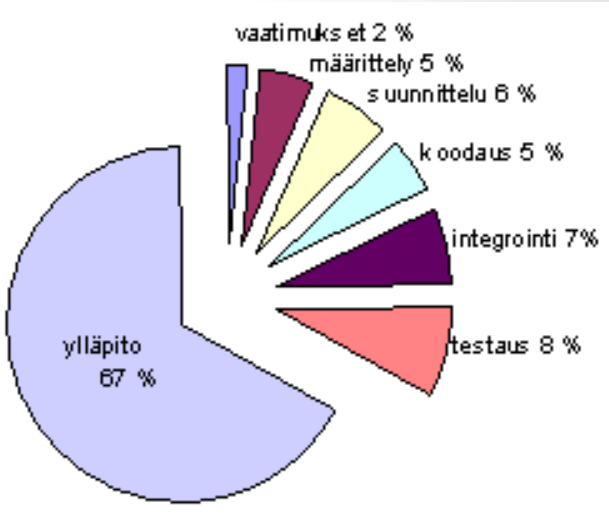
- Teoria / konteksti
- Teoria / käytännönläheinen osuus
- Käytäntö
- Koodiesimerkkejä
- Lopuksi
- Liitteet: Eri tyypillisiä virheitä ja ideoita testaamiseen

Teoriaa / konteksti

Ohjelmistojen elinkaarikustannukset
Laadunvarmistus
Koodin tarkastus
Erilaisia virheitä

Ohjelmistojen elinkaarikustannukset

- Ohjelmistojen koko elinkaaren (määrittely.. toteutus.. käytön lopetus) aikaiset kustannukset koostuvat monista erilaisista osista
- Ohjelmointi on noin 5 % kokonaiskustannuksista
- Ylläpito on noin 2/3 kokonaiskustannuksista
  - Kannattaa muistaa kommentit…



Laadunvarmistus

- **Laadunvarmistuksen** tavoitteena on yleisesti ottaen
  1. Estää virheiden syntyminen
  2. Korjata syntyneet virheet
  3. Arvioida tehdyn tuotteen laatua
- Laadunvarmistus tapahtuu yleensä seuraavilla tavoilla
  1. Koodin lukeminen ja **tarkastaminen** (1-N henkilöä) – vrt. L08T5 tarkastus
  2. **Analysointi** erilaisilla ohjelmilla – vrt. ASPA
  3. Ohjelman suorittaminen eli **testaaminen**
- **Testaaminen** määritellään yleensä tarkoittamaan virheiden etsimistä tietokoneohjelmista niitä **suorittamalla**
- Empiiristen kokeiden perusteella koodin tarkastaminen/lukeminen on testaamista tehokkaampi tapa virheiden poistamiseen

Koodin tarkastus

- Videotykille tulee alla olevan määrittelyn mukainen ohjelma:
  - Tee ohjelma, jonka tulostaa toiminnot sisältävän valikon, kysyy valinnan ja tekee sitten halutun toiminnon. Ohjelma toistaa tätä kunnes käyttäjä lopettaa ohjelman. Valikko on seuraava:
    1) Kirjoita tiedosto
    2) Lue ja tulosta tiedosto
    0) Lopeta
  - Valinnan 1 jälkeen ohjelma kysyy käyttäjältä tiedoston nimen ja kirjoittaa siihen 10 lukua
  - Valinnan 2 jälkeen ohjelma lukee tiedoston ja tulostaa sen sisällön näytölle
  - Valinnan 0 jälkeen ohjelman suoritus päättyy
- Tutustu ohjelmankoodiin ja etsi mahdolliset virheet siitä lukemalla koodia

Huomioita koodin tarkastuksesta

- Suurin osa virheistä löytyi koodia lukemalla eli **tarkastamalla**
- Vaikka ajettavassa ohjelmassa ei ollut virheitä (?) se kuitenkin kaatui käytön aikana käyttäjän antamiin syötteisiin
- Ajonaikaisia *virheitä ei voida aina estää*, mutta niihin voidaan varautua **poikkeustenkäsittelyllä** (exception handling)
  - Poikkeustenkäsittely tarkoittaa sitä, että ohjelmassa varaudutaan normaalista suorituksesta poikkeaviin tilanteisiin
  - Tällä tavoin voidaan ennakoida ja *estää ajonaikaisten virheiden aiheuttamia ongelmia*
- Ohjelmissa voi olla **erilaisia virheitä**

Erilaisia virheitä

- Virheitä voidaan luokitella eri tavoin esim. sen mukaan, missä vaiheessa ja miten ne ovat syntyneet
  1. Kirjoitus- ja syntaksivirhe
     - Tulkki/kääntäjä lopettaa ohjelman suorituksen
  2. Suunnitteluvirhe
     - Ohjelma toimii väärin tai kaatuu ajon aikana virhekohdassa, esim. nollalla jako
  3. Määrittelyvirhe
     - Ohjelma toimii väärin (tai kaatuu ajon aikana virhekohdassa)
- Näitä erilaisia virhetyyppejä on kuvattu tarkemmin Liitteessä 1

Teoriaa / käytännönläheisesti

Testaus – virheiden etsiminen ohjelmasta sitä suorittamalla

Yleistä testaamisesta

- Testaamisen tavoite on löytää virheitä eli onnistunut testi löytää virheen
- Ohjelmaa ei voida osoittaa virheettömäksi testaamalla
  - Voidaan sanoa, että ohjelma toimi oikein annetuilla syötteillä käytetyssä ympäristössä jne.
  - Esim. loppukäyttäjällä oleva virustorjuntaohjelmisto voi estää asennusohjelman toiminnan
- Täydellinen testaaminen ei ole käytännössä mahdollista
- Testaamista varten on usein kirjoitettava erikseen koodia
  - Extreme Programming (XP) menetelmässä kirjoitetaan usein yhtä paljon koodia testaamista varten kuin varsinaista ohjelmakoodia
  - Testauksessa käytettävillä apuohjelmilla on erilaisia rooleja ja nimiä, esim. **testiajuri** ja **testitynkä**
- Testaajat ovat ohjelmoijien työtovereita
  - Omaa koodia ei voi testata kunnolla, sillä omille virheille sokeutuu nopeasti
  - Esimerkiksi kahta koodaajaa kohti voi olla yksi testaaja

Testaustekniikoita

- Positiivinen testaaminen
  - Ohjelma toimii oikein annetuilla syötteillä
- Negatiivinen testaaminen
  - Ohjelma osaa käsitellä virhetilanteet oikein
- Raja-arvot ja testiaineiston luokittelu
  - Raja-arvot: nolla, pienin, suurin, jne.
  - Luokittelu: luvalliset ja luvattomat arvot, positiiviset ja negatiiviset syötteet, eri muuttujatyypit
- Tarkastuslista
  - Ohjelmissa on usein samanlaisia virheitä, joten voidaan muodostaa lista asioista, jotka tulee tarkistaa aina tarkastusten ja testauksen yhteydessä

- Näistä on käytännöllistä tietoa Liitteessä 2

Testiaineisto

- Ohjelman testaaminen tarkoittaa ohjelman suorittamista ja kolmen eri asian seuraamista
  1. Mitä syötteitä ohjelma saa
  2. Mitä ohjelma tekee/tulostaa
  3. Mikä on ohjelman sisäinen tila eli muuttujien arvot
- Käytännössä tarvitaan testiaineistoa, jonka käyttäytyminen tunnetaan, esim.
  - Syötteet ovat 2 ja 3, operaatio "+" ja tulos 5
  - Syöte on *www.maija-mehilainen.fi* ja tulos *maija-mehilainen*
- Testiaineisto == etukäteen dokumentoidut syötteet ja niitä vastaavat tulokset
- Suorituksen jälkeen *saatua tulosta* voidaan verrata *odotettuun tulokseen* ja päätellä, toimiko ohjelma oikein vai väärin näillä syötteillä
  - Huom. ***Näillä syötteillä*** ja ***tällä kertaa***

Virheiden paikallistaminen

- Virheen tunnistamisen jälkeen on löydettävä paikka, missä virhe tapahtuu
  - Virhe voi näkyä tulosteissa ohjelman lopussa, mutta itse virhe on kenties tehty ohjelman alussa syötteiden kanssa
- Tyypillisiä tapoja virheiden tunnistamiseen ja paikallistamiseen ovat mm.
  - **Testitulosteet**: laitetaan koodiin ylimääräisiä tulosteita, joilla pyritään paikallistamaan virheen syntykohta, tyypillisesti jokaisen aliohjelman alussa/lopussa tms.
  - **Debuggeri**: erillinen ohjelma, jolla voi seurata ohjelman suoritusta ja katsoa minkälaisia arvoja muuttujilla on eri kohdissa ohjelmaa suoritusaikana
- Tyypillisesti virhettä etsittäessä se pyritään eristämään siten, että tiedetään varmasti, mikä kohta ohjelmasta toimii oikein ja mikä ei. Usein etsintä etenee aliohjelma kerrallaan ohjelman suoritusjärjestyksessä tavoitteena löytää se aliohjelma, jossa virhe tapahtuu. Sitten käydään epäilty aliohjelma läpi koodilohko ja rivi kerrallaan kunnes virheellinen toiminto ja sen aiheuttama kohta löytyy

Teoriaa / käytännönläheisesti

Poikkeustenkäsittely eli virhetilanteet käsittelevän koodin lisääminen ohjelmaan

Poikkeustenkäsittely

- Poikkeustenkäsittelyn suunnittelussa lähtökohtana on
  1. tunnistaa koodin osa, missä poikkeus voi tapahtua
  2. tunnistaa, mitä poikkeuksia koodissa voi tapahtua
  3. määritellä, miten eri poikkeusten tapahtuessa toimitaan
- Tämän jälkeen voidaan tehdä ohjelmakoodi, joka huolehtii ohjelman suorittamisesta halutuissa virhetilanteissa. Tämän tavoite voi olla
  1. virheestä **toipuminen** ja ohjelman **jatkaminen**
  2. ohjelman suorittamisen **hallittu lopetus**

Poikkeukset

- Pythonin poikkeukset on esitelty Python-dokumentaatiossa **The Python Standard Library** luvussa **Built-in Exceptions**
- Tällä kurssilla tyypillisiä poikkeuksia ovat
  - Exception, ValueError, TypeError, NameError, IndexError, KeyboardInterrupt, SystemExit, FileNotFound, OSError

Käytäntö

Kurssin tehtävien teko ja testaus

Poikkeustenkäsittely

Kurssin tehtävien teko ja testaus 1/2

- Yleistä
  - Ohjelmointi on negatiivisesti ajatellen ongelmasta ongelmaan menemistä, mutta positiivisesti ajatellen uusia haasteita riittää kaikille kokemuksesta riippumatta
  - Aina on tehtäviä, joita joutuu miettimään kaksi tai kolme kertaa ennen kuin homma onnistuu
  - Piirtäminen paperille on yksi hyvä tapa hahmottaa asioita
- Tee ensin kaikki perustoiminnot ja **varmistu niiden oikeasta toiminnasta**, esim.
  - tiedoston luku/läpikäynti (ja testituloste näytölle)
  - arvojen sijoittaminen olioon (ja testituloste näytölle – huom. jäsenmuuttujien arvot, ei olion osoite)
  - olioiden/tiedon laittaminen listaan (ja listan läpikäynti sekä testituloste näytölle)
  - valintarakenteet (ja kaikkien valintojen toiminnan testaus)
  - tulosteet/tallenteet halutussa muodossa (odotettu tulos ja saatu tulos)
- Jätä soveltava osuus viimeiseksi mahdollisuuksien mukaan, esim. pienimmän löytäminen, tietojen haastavampi valinta tms.

Kurssin tehtävien teko ja testaus 2/2

- Kysyä neuvoa keskustelupalstoilla tai käy neuvontatilaisuuksissa
  - **Tausta**: ”Saan tehtyä x ja y:n”, ”Ymmärrän tämän asian” - ja osoita, ettei näissä ole ongelmaa
    - Testaamalla voidaan osoittaa, ettei yksittäisessä kohdassa ole ongelmaa annetulla aineistolla
  - **Ongelma**: ”En ymmärrä”, ”En saa otettua/tehtyä/laskettua”, … - ongelman tunnistaminen on keskeinen asia
    - Kuva ongelmasta helpottaa tilanteen ymmärtämistä
  - **Varsinainen kysymys**: Miten pääsen eteenpäin – voit esittää vaihtoehtoja, kumpi kannattaa jne.
- Keskustelupalstoilla kannattaa pysyä samassa kysymyksessä siihen asti, että tehtävä tulee ratkaistua. Lauseen tai kahden yhteenveto ongelman ratkeamisesta on arvokas muille saman ongelman parissa painiville eli ratkesiko ja miten ratkesi
- Vinkkejä tämän kurssin ohjelmien testaamiseen löytyy Liite 2:ssa

Poikkeustenkäsittely tällä kurssilla

- Tällä kurssilla tavoitteena on ymmärtää, miten poikkeustenkäsittely toimii ja pystyä käsittelemään poikkeukset aina tiedostonkäsittelyn yhteydessä eli tiedostoa avattaessa, luettaessa tai kirjoitettaessa. Näiden operaatioiden tulee olla saman try-except –rakenteen sisällä
- Poikkeuksista ei tarvitse tällä kurssilla toipua hallitusti vaan riittää, että käyttäjälle kerrotaan mitä ja missä tapahtui ja lopetetaan ohjelma hallitusti (sys.exit(0) –käskyllä)
  - Poikkeustenkäsittely tehdään Exception-poikkeuksen avulla
  - Unix tarkistaa ohjelmien paluuarvot, joten palauta ohjelmasta aina 0 sen lopuksi – myös CodeGrade tarvitsee paluuarvona 0:n
- Harjoittelumielessä ja ymmärtämisen kannalta poikkeustenkäsittely käydään läpi tiedostonkäsittelyn ja käyttäjäsyötteiden näkökulmista
- ***Harjoitustyössä ja tentissä on oltava poikkeustenkäsittely tiedostonkäsittelyn yhteydessä***
  - ***Katso Koodiesimerkkejä-kalvot***

Virheestä toipuminen ja tarvittava lisäkoodi

- Ohjelma yrittää toipua virheestä try…except rakenteella, esimerkiksi

```
Luku=1
Jakaja = 0
try:
      Luku1 = Luku / Jakaja # nollalla jako
except ZeroDivisionError:
      Jakaja = int(input("Nolla jakajana, anna parempi"))
      Luku1 = Luku / Jakaja
      print(Luku1)
# Ohjelman normaali suoritus jatkuu tästä
```

- Huom. try-except avainsanojen välissä tarkkailtava koodi rajoittuu vain oleellisiin käskyihin
- Oikeasti toimiva virheestä toipuminen edellyttää usein isompia rakenteellisia muutoksia ohjelmaan
  - Esim. yo. esimerkissä käyttäjän antama toinen nolla kaataa ohjelman
- Useita poikkeuksia voi yhdistää pilkuilla laittamalla lista sulkuihin ”except (OSError, ValueError, TypeError):”
- Useita poikkeuksia voi erotella laittamalla ne allekkain ohjelmaan, esim. alkuun voi nimetä tietyt poikkeukset sekä antaa niille omat jatkotoimenpiteet ja viimeisenä laittaa muille poikkeuksille yhteisen käsittelyn

Ohjelman hallittu lopettaminen

- Ohjelman hallittu lopettaminen voidaan tehdä `try…finally` rakenteella eli esimerkiksi

```
import sys
Luku = 1
Jakaja = 0
try:
    Luku1 = Luku / Jakaja # nollalla jako
finally:
    print("Lopetetaan ohjelman suoritus")
    sys.exit(0)
```

Virheenkäsittely laajemmin

- Olio-ohjelmoinnissa virheenkäsittely hoidetaan pääasiassa poikkeustenkäsittelynä
- Virheitä ja ongelmia voidaan ennakoida ja estää myös normaalilla valintarakenteella tyyliin

```
if (Jakaja == 0):
    print(”Jakaja nolla…”)
    …
```

- Erilaiset virheenkäsittelytekniikat sopivat erilaisiin tilanteisiin, joten jokaiseen tilanteeseen kannattaa katsoa sopiva toimintatapa
  - Tällä kurssilla tavoite on ymmärtää virheenkäsittely ja poikkeustenkäsittely peruskonsepteina, joiden tarkempi opettelu tapahtuu myöhemmin

Koodiesimerkkejä

Tiedostojen poikkeustenkäsittely

Valikkopohjainen ohjelma virheenkäsittelyllä

Tiedostojen poikkeustenkäsittely

- Alla tämän kurssin minimaalinen poikkeustenkäsittely

```
import sys
Nimi = "data.txt"

try:
    Tiedosto = open(Nimi, 'r')
    Rivi = Tiedosto.readline()
    print(Rivi, end='')
    Tiedosto.close()
except Exception:
    print("Tiedoston '{0:s}' käsittelyssä virhe, lopetetaan.".
                                                    format(Nimi))
    sys.exit(0)
```

Valikkopohjainen ohjelma virheenkäsittelyllä

```
import L09DemoKirjasto

def paaohjelma():
    Valinta = 1
    TiedostoLue = "L06Lue.txt"
    TiedostoKirjoita = "L09Kirjoita.txt"
    ListaSyote = []
    ListaTulos = []
    IndeksiMax = None
    Indeksi = 0
    while (Valinta != 0):
        Valinta = L09DemoKirjasto.valikko()
        if (Valinta == 1):
            if (IndeksiMax == None):
                ListaSyote = L09DemoKirjasto.lueTiedosto(TiedostoLue, ListaSyote)
                IndeksiMax = len(ListaSyote) - 1
            else:
                Indeksi += 1

            if (Indeksi <= IndeksiMax):
                Merkkijono = ListaSyote[Indeksi]
            else:
                print("Merkkijonot loppuivat, lopeta ohjelma.")
        elif (Valinta == 2):
            ListaTulos.append(Merkkijono)
            print("Lisätty listaan merkkijono '" + Merkkijono + "'.")
        elif (Valinta == 3):
            Merkit = Merkkijono[::-1]
            ListaTulos.append(Merkit)
            print("Lisätty listaan merkkijono '" + Merkit + "'.")
        elif (Valinta == 0):
            print("Lopetetaan.")
        else:
            print("Tuntematon valinta, yritä uudestaan.")
        print()
    L09DemoKirjasto.tallennaTiedosto(TiedostoKirjoita, ListaTulos)
    ListaSyote.clear()
    ListaTulos.clear()
    print("Kiitos ohjelman käytöstä.")
    return None

paaohjelma()
```

```
# 20221107 L09DemoKirjasto.py un Laajeneva demo: virheenkäsittely
import sys

def valikko():
    print("1) Lue merkkijono")
    print("2) Lisää listaan merkkijono etuperin")
    print("3) Lisää listaan merkkijono takaperin")
    print("0) Lopeta")
    Syote = input("Anna valintasi: ")
    Valinta = int(Syote)
    return Valinta

def lueTiedosto(Nimi, Lista):
    try:
        Tdsto = open(Nimi, "r", encoding="UTF-8")
        Rivi = Tdsto.readline()[:-1]
        while (len(Rivi) > 0):
            Lista.append(Rivi)
            Rivi = Tdsto.readline()[:-1]
        Tdsto.close()
    except Exception:
        print("Tiedoston '{0:s}' käsittelyssä virhe,  lopetetaan.".format(Nimi))
        sys.exit(0)
    Tulosta = "Luettu tiedosto '" + Nimi + "'."
    print(Tulosta)
    return Lista

def tallennaTiedosto(Nimi, Lista):
    try:
        Tdsto = open(Nimi, "w", encoding="UTF-8")
        for Str in Lista:
            Rivi = Str + '\n'
            Tdsto.write(Rivi)
        Tdsto.close()
    except Exception:
        print("Tiedoston '{0:s}' käsittelyssä virhe, lopetetaan.".format(Nimi))
        sys.exit(0)
    Tulosta = "Tallennettu tiedosto '" + Nimi + "'."
    print(Tulosta)
    return None
```

Lopuksi

Osaamistavoitteet

Ohjelmointivideot – Video 2 debuggaus

Osaamistavoitteet

- Teoria/konteksti
  - Ohjelmistojen elinkaarikustannukset, laadunvarmistus, tarkastaminen, erilaisia virheitä
- Teoria/käytännönläheisesti
  - Testaamisesta – virheiden etsiminen valmiista ohjelmasta
  - Poikkeustenkäsittely – virhetilanteita käsittelevän koodin lisääminen ohjelmaan
- Käytäntö
  - Oman ohjelman testausta eri näkökulmista
  - Poikkeustenkäsittely – oltava aina tiedostonkäsittelyn yhteydessä
  - Debuggaus – tällä kurssilla ei tarvitse käyttää debuggeria, testitulosteet hyvä lähtökohta. IDLEssä on yksinkertainen debuggeri, jota kannattaa kokeilla

Video 2: Python debuggerin käyttöohje

- Aja ohjelmaa L09Debugger.py debuggerin kanssa
  - Debuggerilla voi suorittaa ohjelmaa yksi askel kerralla, siirtyä halutulle riville, tutkia muuttujien arvoja ja lopettaa ohjelman suorituksen
  - Debugger ei muuta ohjelman suoritusjärjestystä
- Oleellisia toimintoja
  - Python Shell: Debug | Debugger päälle (check-merkki)
  - Lähdekoodi ja hiiren oikea nappi: Set/Clear Breakpoint
  - Debug Control –ikkuna
    - Muuttujien arvot näkyvät ikkunan alaosassa
    - Go: suorittaa ohjelman seuraavaan breakpoint'iin tai loppuun asti
    - Step: yksi askel eli käsky kerrallaan
    - Over: aliohjelma suoritetaan yhtenä askeleena
    - Out: suoritetaan aliohjelma loppuun ja siirrytään kutsua seuraavaan käskyyn
    - Quit: lopeta debuggaus ja ohjelman suoritus

Täydennyksiä oppaan lukuun 9

Tyyliohjeita pienille Python-ohjelmille
ASPAn tarkistukset
Oppaan esimerkit ja käsitellyt asiat

Pienen Python-ohjelman tyyliohjeet

- Poikkeustilanteet käsitellään poikkeustenkäsittelyllä. Tällä kurssilla lähtökohta on
  - try-except –rakenne ja hallittu lopetus sys.exit(0) –käskyllä
- Tällä kurssilla poikkeustilanteet on tunnistettava tiedostonkäsittelyn yhteydessä ja ohjelma tulee lopettaa silloin
- Virhetilanteet kannattaa pyrkiä tunnistamaan ja estämään ennakolta tyypillisesti if-lauseella
  - Tyypillisesti tarkistetaan ennen analyysiä tai kirjoittamista, että tietorakenteessa on dataa näitä toimenpiteitä varten

ASPAn L09 tarkastukset

- Poikkeustenkäsittelyyn liittyen ASPA tarkastaa ohjelmasta seuraavat asiat
  - Poikkeustyypit on nimetty except-komennon jälkeen
  - Kaikki tiedoston avaus-, luku- ja kirjoituskäskyt ovat poikkeustenkäsittelyn sisällä
  - Poikkeustenkäsittely on toteutettu samassa aliohjelmassa tiedostonkäsittelyn kanssa, ts. ei kutsuvassa ohjelmassa
- Muista, että vain poikkeukseen suoraan liittyvä koodi on poikkeuksenkäsittelijän sisällä, ts. koko aliohjelmaa ei laiteta poikkeustenkäsittelyn sisään. Tämä asia tarkistetaan assistentin toimesta

Käsitellyt asiat oppaan luvussa 9

- Poikkeustenkäsittely, try-except: Esimerkki 9.2, 9.3, 9.5
- Poikkeustenkäsittely, try-finally: Esimerkki 9.4
- Virhetilanteiden ennaltaehkäisy: Esimerkki 9.1
  - Huom. Tarkistus kannattaa tehdä kutsuvassa aliohjelmassa luentoesimerkkien mukaisesti

- **Huom. Kokoava esimerkki 9.5, tällä kurssilla tiedostonkäsittelyn yhteydessä vaadittava poikkeustenkäsittely**

Liitteet

1. Erityypillisiä virheitä
2. Ideoita testaamiseen

Virhetyypit 1: Kirjoitus- ja syntaksivirheet

- Esimerkki kirjoitusvirheestä
  - Print kun pitäisi olla print
  - Tyypillisesti Python-tulkki antaa SyntaxError –virheilmoituksen
- Syntaktisesti eli kieliopillisesti väärin
  - a = 3 – oikein
  - **b = (a = 3) – väärin**
  - Python ei hyväksy b:n arvoksi lauseketta "(a=3)"
  - Seuraavat lauseet ovat syntaktisesti oikein
    - b = a = 3
    - b = (a == 3)

Virhetyypit 2: Suunnitteluvirhe

- Esimerkki suunnitteluvirheestä on alkuehtoinen toistorakenne, jossa lopetusehto ei toteudu koskaan:

```
a = 1
while (a < 5):
    print(a)
```

- Suunnitteluvirhe on yleensä tilanne, joka ohjelmoijan olisi pitänyt huomata ja käsitellä, mutta jostain syystä näin ei ole tehty, esim.
  - Tiedoston avaaminen ei onnistu, koska tiedostoa ei ole
  - Tiedostoon kirjoittaminen ei onnistu, koska levy on täysi tai kirjoitussuojattu
  - Näihin erikoistilanteisiin varautuminen on suunnittelua (poikkeustenkäsittely)

Virhetyypit 3: Määrittelyvirhe

- Määrittelyvirhe tarkoittaa esim. sitä, että ohjelmasta puuttuu jokin keskeinen toiminnallisuus/asia
- Lähtökohtaisesti käyttäjän pitäisi tietää, mitä hän tarvitsee
  - Käyttäjän pitäisi aktiivisesti kertoa tarpeistaan
  - Käyttäjän pitäisi osata vastata ohjelman määrittelijän tekemiin kysymyksiin ohjelmassa tarvittavista ominaisuuksista
- Käytännössä määrittelijän pitäisi sopia ja dokumentoida yhdessä käyttäjän kanssa ohjelman toiminta kaikissa tapauksissa, niin normaali- kuin poikkeustapauksissa

Testaus 1: Oman ohjelman testaus

- Ensimmäinen vaihe on varmistaa, että ohjelma toimii tehtäväksiannon mukaisesti:
  - Suorita ohjelma tehtävänantoa seuraten ja tarkista, että valmis ohjelma toimii juuri niin kuin tehtäväksiannossa lukee
  - Ohjelma ei saa kaatua missään vaiheessa ja tulosten tulee olla tehtäväksiannon mukaisia
- Tätä vaihetta kutsutaan **positiiviseksi testaukseksi**, ts. kaikki toimii hienosti ilman virheitä

Testaus 2: Negatiivinen testaus

- Toinen vaihe on varmistua, että ohjelma pystyy käsittelemään virhetilanteet hallitusti. Tätä voidaan testata **negatiivisella testauksella**
  - Katso tehtäväksiannosta ohjelmalle annettavat syötteet ja niiden rajaukset. Sen jälkeen anna ohjelmalle syötteitä, jotka eivät ole näiden rajausten mukaisia
    - Esimerkiksi jos ohjelma odottaa kokonaislukua, anna sille desimaaliluku, kirjainmerkki tai erikoismerkki ts. väärän tyyppinen tietoalkio
    - Kokeile suorittaa ohjelma antamatta pyydettyjä syötteitä
    - Jos syötteillä on minimi- tai maksimirajoja (arvo, pituus, tms.), käytä syötteitä, jotka ovat hyväksyttävien rajojen ulkopuolella
  - Edelleenkään ohjelma ei saisi kaatua ja tulosten tulisi olla tehtäväksiannon mukaisia ja tyypillisesti virheilmoituksia

Testaus 3: Tarkastuslista

- Ohjelmissa on usein samanlaisia virheitä, joten tarkastuslista auttaa poistamaan ne. Tyypillisiä kohtia tarkastuslistalla ovat mm.
  - Raja-arvot: varmistu, että ohjelma osaa käsitellä dokumentaation mukaiset raja-arvot oikein – raja-arvo, sitä edellinen sekä sitä seuraava arvo
    - Silmukat ja listat: tyypillinen raja-arvoesimerkki eli varmista, että ohjelman silmukat käyvät läpi oikein ensimmäisen, toisen, toiseksi viimeisen ja viimeisen arvon
  - Nollalla jako: varmistu, ettei ohjelmassa tapahdu missään vaiheessa nollalla jakoa tai sellaiset kohdat ovat poikkeustenkäsittelyn sisällä
  - Virheenkäsittely: varmistu, että ohjelmassa on virheenkäsittelijät ainakin tiedoston avaamisen, lukemisen ja kirjoittamisen yhteydessä
- ***Huomaa***
  - L08T5 ja harjoitustyön tarkastuslista – tekijä voi tarkastaa asiat ennen palautusta

Testaus 4: Valikkopohjaisen ohjelman testaaminen

- Tyypillisiä testattavia kohtia
  - Valikon tulostus: helpointa hallita aliohjelmana, tulostuuko oikein
  - Valinnan kysyminen: tietotyyppi, onko paluuarvo oikein
  - Valintarakenteen läpikäynti: kaikkien vaihtoehtojen läpikäynti, käyttäjän syöte ja valintarakenteen testit samoilla tietotyypeillä, luvattomien syötteiden käsittely
  - Valintarakenteesta oikeisiin toimintoihin: helpointa kutsua valikosta sopivia aliohjelmia, päädytäänkö kaikkiin aliohjelmiin, palataanko kaikista aliohjelmista hallitusti takaisin

Testaus 5: Tiedostonkäsittelyn testaaminen

- Tiedostonkäsittelyn testauksessa joudutaan usein käyttämään apuohjelmia tiedostojen avaamiseen ja tutkimiseen tehtävän ohjelman ulkopuolella. Tyypillisiä työkaluja esim. tiedostonhallinta ja tekstieditori
- Tiedoston avaaminen: onnistuuko ja onko virheenkäsittely
- Tiedoston kirjoittaminen: onnistuuko, meneekö tiedot oikein tiedostoon (teksti-/binaaritiedosto), meneekö loppumerkki oikein, kirjoitetaanko jatkoksi vai tuhotaanko entinen tiedosto
- Tiedoston lukeminen: onnistuuko, onko tiedostomuoto oikea (teksti/binaari; tiedoston rivirakenne), päättyykö tiedosto odotetusti, onko tiedostossa odotettua tietoa/tietomuotoja
- Onnistuuko erikokoisten tiedostojen käsittely: tyhjä tiedosto (luo/tyhjennä), 0 riviä, 1 rivi, monta riviä, paljon rivejä esim. 2 000 000 riviä…

Testaus 6: Aliohjelmien testaaminen

- Aliohjelmakutsu: meneekö aliohjelmaan ja palaako sieltä suunnitellusti
- Parametrit ja paluuarvot: menevätkö halutut tiedot aliohjelmaan ja saadaanko sieltä odotetut tiedot takaisin, erityisesti tietotyypit ja lukumäärät
- Tekeekö aliohjelma odotetut toiminnot oikein: testiaineisto sisältää syötteet ja odotetun tuloksen eli laskeeko oikein tai muuttaako esim. tiedoston sisältöä halutulla tavalla jne.