LUT Python ohjelmointiopas 2023 - osa 13 Sisällysluettelo

Luku 13: Käyttöliittymistä

Tietokoneiden ohjelmointia tehtiin vielä 1960 luvulla reikäkorttien avulla. Reikäkorttiin perustuva automaattinen tietojenkäsittely kehitettiin Yhdysvaltain väestölaskentaa varten 1890 kutomakoneiden reikäkorttiohjauksen pohjalta. Automaattinen tietojenkäsittely näytti siinä määrin lupaavalta alalta, että sitä varten perustettiin yhtiö, jonka nimi oli vuodesta 1924 alkaen IBM. Alkuaikoina reikäkortteja käytettiin tiedon koodaamiseen eli tieto, numerot ja merkit, lävistettiin kartongista valmistettuun pahviin reikinä, joiden paikka määritteli halutun tiedon (http://suomentietokonemuseo.fi/vanhat/fin/laite_fin.htm). Reikäkortteja käytettiin siis alussa kutomakoneiden ohjaukseen ja tiedon tallettamiseen, mutta myöhemmin myös tietokoneohjelmia koodattiin reikäkorteilla.

Siirtyminen reikäkorteista ohjelmien koodaamiseen näppäimistön avulla helpotti työtä merkittävästi. Tosin koodaaminen ilman näyttöä oli haastavaa ja se tehtiin rivieditorilla, jossa merkkejä voitiin syöttää yhdelle riville kerrallaan syöttömoodissa tai rivejä voitiin muokata kokonaisina riveinä komentomoodissa. Nykyään koko ohjelma on tyypillisesti nähtävillä näytöllä, mutta rivieditoreita löytyy vielä. Esimerkiksi Unix/Linux käyttöjärjestelmissä on tyypillisesti mukana rivieditori ed, josta kehitettiin näyttöjä hyödyntäviä editoreita kuten vi (visual editor) ja myöhemmin emacs.

Näppäimistön käyttö muutti ohjelmointia merkittävästi ja vaikka graafiset käyttöliittymät vallitsevat nykyään markkinoita, ohjelmat perustuvat vielä tyypillisesti tekstitiedostoihin. 1970-luvulla kehitetty Unix-järjestelmä perustuu merkkipohjaisiin työkaluohjelmiin, vaikka nykyään useimmat niistä ovat saaneet graafisia käyttöliittymiä tai niihin perustuvia seuraajia. Merkkipohjaisille käyttöliittymille on kuitenkin vielä oma käyttäjäkuntansa ja sulautetuissa järjestelmissä on usein paljon koodia ilman käyttäjälle näkyvää käyttöliittymää. Esimerkiksi auton lukkiutumattomat jarrujärjestelmät perustuvat ohjelmistoihin, vaikkei käyttäjä pääsekään vaikuttamaan niihin oman käyttöliittymän kautta.

Tässä luvussa esittelemme lyhyesti graafisten käyttöliittymien perusteet ja perinteisen komentorivikäyttöliittymän. Katsomme ensin miltä graafiseen käyttöliittymään perustuva ohjelma näyttää Pythonilla tehtynä. Sen jälkeen tutustumme lyhyesti komentorivikäyttöliittymään ja miten komentoriviparametrit saadaan hoidettua Pythonilla.

Graafisten käyttöliittymien perusteet

Nykyisin useimmat ohjelmat julkaistaan graafisiin käyttöliittymiin perustuvissa käyttöjärjestelmissä. Tämän vuoksi käytännössä kaikki laajemmat ohjelmat, ohjelmistot sekä työkalut toimivat nimenomaan graafisella käyttöliittymällä, jossa valinnat ja vaihtoehdot esitetään valikoina tai valintoina, joiden avulla käyttäjä voi hiirellä tai kosketusnäytöllä valita mitä haluaa tehdä. Monesti aloitteleva ohjelmoija luulee, että tällaisen käyttöliittymän toteuttamista varten tarvitaan jokin monimutkainen tai kallis kehitystyökalu tai että se olisi erityisen vaikeaa. Ehkä joidenkin ohjelmointikielien yhteydessä tämä pitää paikkansa, mutta Pythonissa yksinkertaisten graafisten käyttöliittymien tekeminen onnistuu helposti Tkinter-kirjastomoduulin avulla.

Tkinter on alun perin Tcl-nimisen ohjelmointikielen käyttöliittymätyökalusta Tk tehty Python-laajennus, jonka avulla voimme luoda graafisen käyttöliittymän ohjelmallemme. Tkinter-moduuli poikkeaa siinä mielessä ”perinteisistä” Python-moduuleista, että sitä käyttävä Python-koodi poikkeaa hyvin paljon tavallisesta Python-koodista ulkonäkönsä puolesta. Kuitenkin Python-kielen syntaksisäännöt sekä rakenne pysyvät edelleen samoina. Koodi voidaan edelleen tuottaa yksinkertaisesti tekstieditorilla, josta tulkki tuottaa graafisen esityksen. Kannattaa kuitenkin huomata, että laajempia rakenteita sisältävä Tkinter-ohjelmakoodi on käytännössä aina pitkä ja monesti myös melko työläs kirjoitettava, joten koodin ajamista suoraan tulkin ikkunassa ei kannata yrittää. Käytettäessä Tkinter-moduulia

ainoa käytännössä järkevä lähestymistapa on luoda lähdekooditiedosto, joka tallennetaan ja ajetaan sellaisenaan tulkista.

Seuraavilla kahdella esimerkillä tutustumme siihen, kuinka yksinkertaisen graafisen ikkunan tekeminen onnistuu. Aloitetaan tekemällä pohja, johon käyttöliittymä rakennetaan.

Graafinen käyttöliittymä Pythonilla

Esimerkissä 13.1 on minimaalinen graafisen käyttöliittymän luova Python ohjelma. Koodi on lyhyt eikä sisällä paljoakaan toimintoja. Ensimmäisellä rivillä otamme käyttöön Tkinter-kirjaston. Tällä kertaa sisällytys toteutetaan aiemmasta poikkeavalla `from...import *` -syntaksilla johtuen siitä, että se on tämän kirjaston yhteydessä hyvä ratkaisu.

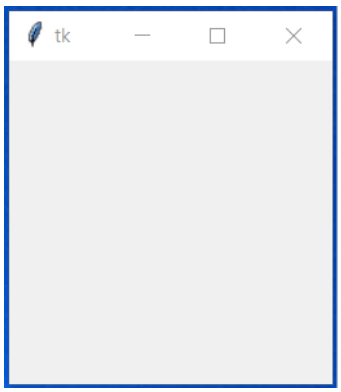
Ohjelman toisella rivillä luomme käyttöliittymän perustan tekemällä muuttujasta `Pohja` Tk()-luokan juuri- eli pohjatason (`root`). Tähän pohjatasoon lisäämme jatkossa kaikki haluamamme komponentit ja toiminnot, mutta tällä kertaa jätämme sen tyhjäksi. Kolmannella rivillä käynnistämme Tkinter-käyttöliittymän kutsumalla pohjatasoa sen käynnistysfunktiolla `mainloop` ja ohjelma lähtee käyntiin tuottaen kuvan 13.1 tyhjän ikkunan. Tyhjä ikkuna johtuu siitä, ettei ohjelman pohjatasolle ole sijoitettu muita komponentteja. Jos olisimme laittaneet sille vaikka painonapin, olisi ruutuun ilmestynyt painonappi.

Esimerkki 13.1. Perusikkuna

```
from tkinter import *

Pohja = Tk()
Pohja.mainloop()
```

Esimerkin 13.1 koodin tuottaa ajettaessa kuvan 13.1 tyhjän käyttöliittymäikkunan.



Kuva 13.1: Esimerkin 13.1. tuottama käyttöliittymäikkuna. Ikkunan ulkoasu vaihtelee käyttöjärjestelmän ja ikkunamanagerin mukaan

Komponenttien lisääminen

Käyttöliittymästä ei ole paljoa iloa, jos siinä ei ole mitään muuta kuin tyhjiä ikkunoita. Tämän vuoksi haluammekin lisätä ruutuun komponentteja (widgets), joiden avulla voimme lisätä ruutuun tarvitsemamme toiminnot. Komponentit lisätään aina joko suoraan pohjatasolle tai vaihtoehtoisesti `frame`-komponenttiin, joka toimii säiliönä ja tilanjakajana Tkinter-käyttöliittymässä. Tarkastelkaamme seuraavaksi, kuinka pohjatasolle lisätään tekstikenttiä ja syöttölaatikoita. Huomaa, että seuraava esimerkki ei ole tämän oppaan ydinasioita, sillä toteutuksessa on käytetty olio-ohjelmointia, jossa on mukana luokan määrittely jäsenmuuttujineen ja -funktioineen. Tämä esimerkki on siis tarkoitettu niille, joita asia kiinnostaa.

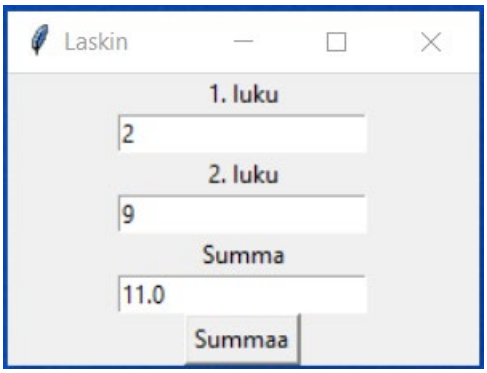
Esimerkki 13.2. Kehittyneempi graafinen ohjelma

```
from tkinter import *

class LASKIN():
    Luku1 = ""
    Luku2 = ""
    Vastaus = ""
    def __init__(self):
        Pohja = Tk()
        Pohja.title('Laskin')
        Label(Pohja, text='1. luku').pack()
        self.Luku1 = Entry(Pohja)
        self.Luku1.pack()
        Label(Pohja, text='2. luku').pack()
        self.Luku2 = Entry(Pohja)
        self.Luku2.pack()
        Label(Pohja,text='Summa').pack()
        self.Vastaus = Entry(Pohja)
        self.Vastaus.pack()
        Button(Pohja,  text='Summaa', command=self.summaa).pack()
    def summaa(self):
        Eka = float(self.Luku1.get())
        Toka = float(self.Luku2.get())
        Summa = Eka + Toka
        self.Vastaus.delete(0, END)
        self.Vastaus.insert(0, str(Summa))

Kalkulaattori = LASKIN()
```

Kun ajamme esimerkin 13.2. koodin, tuottaa tulkki kuvassa 13.2 näkyvän käyttöliittymäikkunan. Tässä lähdekoodissa tuotamme käyttöliittymän, joka tekee jotain. Alkuun suoritamme samanlaiset alustustoimenpiteet kuten aiemmin eli otamme käyttöön Tkinter-moduulin. Seuraavaksi määrittelemme luokan `LASKIN`, joka hoitaa niin laskimen toiminnallisuuden kuin graafisen käyttöliittymänkin.



Kuva 13.2: Esimerkin 13.2 näytölle tuottama ikkuna eli Laskin-ohjelma

Luokalla on kolme jäsenmuuttujaa: `Luku1`, `Luku2` ja `Vastaus`. Aluksi nämä alustetaan tyhjiksi ja luokan alustusvaiheessa näihin sijoitetaan syöttölaatikot. Alustuksen aluksi luodaan jälleen pohja, johon kaikki ohjelmamme komponentit sijoitetaan. Tällä kertaa määrittelemme ikkunalle otsikon ”Laskin”. Seuraavaksi lisäämme tekstikentän ja paketoimme sen samalla ohjelmaan mukaan. Tekstikenttä soveltuu staattisen tekstin esittämiseen ja parametreina annamme pohjan, johon tekstikenttä lisätään ja toisena parametrina itse lisättävän tekstin. Tämän jälkeen sijoitamme jäsenmuuttujaan `Luku1` syöttökentän `Entry(Pohja)` määritteen avulla. Tämä luo laatikon, johon käyttäjä voi syöttää tietoa, jonka jälkeen paketoimme sen mukaan ohjelmaamme. Toistamme samat

vaiheet vielä kahteen kertaan, jotta saamme kaikki tarvittavat komponentit mukaan. Viimeiseksi lisäämme mukaan nappulan (`Button`), jolla itse yhteenlasku tapahtuu. Nappulalle annamme parametrina `Pohjan`, jotta se menee oikeaan paikkaan; tekstin, joka tulee nappulaan; ja kolmantena `command`-parametrina viitteen luokan omaan `summaa` jäsenfunktioon. Näin laskin kutsuu tätä funktiota aina kun nappulaa painetaan.

`summaa` jäsenfunktiossa kysymme luokkamme jäsenmuuttujilta (jotka ovat tyyppiä `Entry`) niiden sisältöä metodilla `get`. Tällä tavoin saatu sisältö on tekstiä, joten se täytyy muuntaa `float`-funktiolla desimaaliluvuksi. Tilapäissäiliöön `Summa` tallennetaan lukujen yhteenlaskun tulos. Tämän jälkeen `Vastaus`-tekstikentän sisältö tuhotaan alusta loppuun ja sen tilalle lisätään merkkijonoksi muutettu yhteenlaskun summa.

Itse ohjelma käynnistyy luomalla uusi olio `LASKIN`-luokasta tiedoston viimeisellä rivillä.

Huomioita Tkinter-moduulista

Koska Tkinter-moduuli on hyvin laaja ja toimintatavoiltaan jonkin verran tavallisesta Python-ohjelmakoodista poikkeava, ei tässä oppaassa käsitellä aihetta tämän enempää. Lisää graafisten käyttöliittymien toteuttamisesta Tkinter-moduulilla voit etsiä Internetistä.

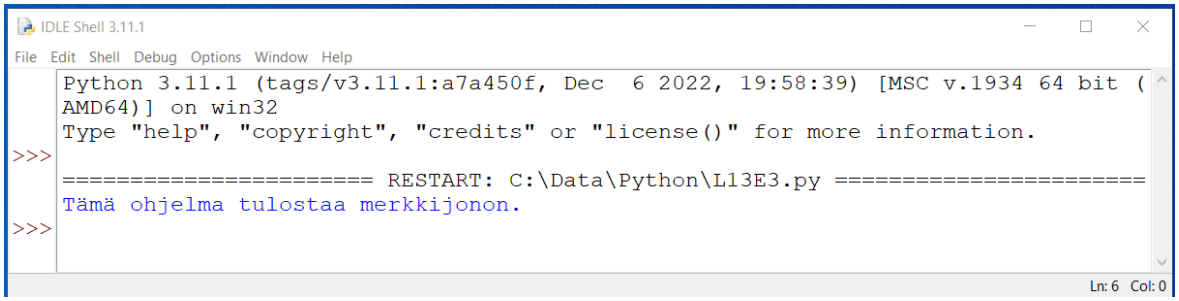
Komentorivikäyttöliittymä ja komentoriviparametrit

Komentorivikäyttöliittymä tarkoittaa sitä, että ohjelma käynnistetään käyttöjärjestelmän komentoriviltä ohjelman nimellä. Tutustutaan asiaan esimerkin 13.3 yksinkertainen ohjelman avulla, jossa on vain yksi tulostuskäsky.

Esimerkki 13.3. Yksinkertainen ohjelma

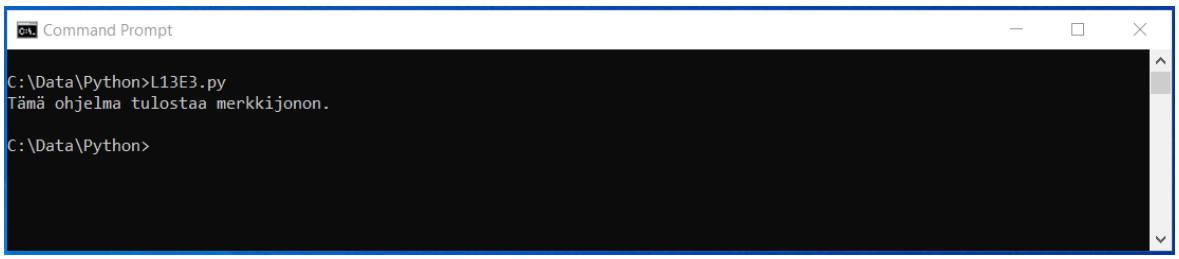
```
print("Tämä ohjelma tulostaa merkkijonon.")
```

Kun esimerkin 13.3 ohjelma suoritetaan normaalisti IDLEn editori-ikkunasta, aukeaa interaktiivinen komentorivitulkki-ikkuna, johon tulee kuvan 13.3 tulosteet.



Kuva 13.3: Esimerkin 13.3 ohjelma suoritettuna IDLE-editorista

Komentorivikäyttö edellyttää siirtymistä Windows'n komentoriville eli avaa komentorivi-ikkuna, cmd.exe, ja siirry kansioon, jossa suoritettava tiedosto on. Nyt suoritettava ohjelma näkyy esimerkissä 13.3 ja se on tallennettu nimellä L13E3.py. Suoritetaan tämä ohjelma komentoriviltä kuvan 13.4 mukaisesti eli kirjoitetaan ohjelman nimi komentorivikehotteen perään ja painetaan enter'ä, jolloin ohjelma tulostaa koodin mukaisen tekstin näytölle.



Kuva 13.4: Esimerkin 13.3 ohjelma suoritettuna komentoriviltä

Nyt kun osaamme suorittaa ohjelman komentoriviltä eli käyttää komentoriviä, voimme tutustua komentoriviparametreihin, joilla voidaan viedä tietoa käynnistettävälle ohjelmalle. Komentoriviparametrit noudattavat samaa ideaa kuin aliohjelman parametrit. Aliohjelmalle parametrit annetaan nimen perässä olevissa suluissa, esim. `print("eka", 2)`, jolloin parametreina ovat merkkijono `eka` ja luku `2`. Jos suoritamme ohjelman IDLEssä, emme saa välitettyä sille tietoa sen käynnistyshetkellä, mutta ohjelma voi kysyä tietoa käyttäjältä input-käskyillä tai lukea tietoa tiedostosta. Komentoriviparametrit mahdollistavat tiedon välittämisen ohjelmalle sen käynnistyshetkellä komentoriviltä.

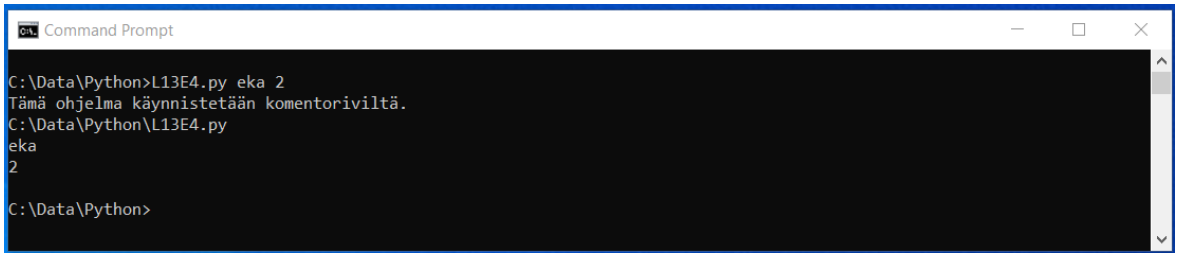
Esimerkissä 13.4 näkyy ohjelma, joka tulostaa kaikki komentoriviparametrillä olleet parametrit näytölle. Ohjelma poikkeaa esimerkistä 13.3 siinä, että se sisällyttää `sys`-kirjaston ja sen jälkeen käy läpi `sys.argv`-listan läpi tulostaen kaikki listan alkiot omille riveilleen kuvan 13.5 mukaisesti. Käytännössä käyttöjärjestelmä ottaa komentoriville kirjoitetut merkit talteen yhtenä merkkijonona ja jakaa sen välilyönneistä osiin, jotka tallennetaan `sys.argv`-listan alkioiksi. Pythonissa listan läpikäynti onnistuu `for`-lauseella, jolla saamme tulostettua kaikki komentorivillä olleet parametrit näytölle merkkijonoina.

Esimerkki 13.4. Komentorivillä olevat parametrit tulostava ohjelma

```
import sys

print("Tämä ohjelma käynnistetään komentoriviltä.")
for Parametri in sys.argv:
    print(Parametri)
```

Komentoriviparametrien osalta on hyvä muistaa, että `argv`-listalla on kaikki komentoriviparametrit siten, että ensimmäisenä oleva ohjelman nimi sisältää myös sen polun ja kaikki parametrit ovat merkkijonoja. Parametrit tulee siis muuttaa sopivaksi tietotyypiksi ennen käyttöä virheentarkistuksia unohtamatta. Mutta tästä eteenpäin ohjelmointi on tuttua asiaa, ja tässä esimerkissä uutta on vain komentoriviparametrien idea sekä `sys`-kirjaston `argv`-listan käyttö.



Kuva 13.5: Esimerkin 13.4 ohjelma suoritettuna komentoriviltä komentoriviparametrit tulostettuina allekkain yksi parametri aina riville

Jos edellä oleva ohjelma ei toimi komentorivillä, kannattaa tarkistaa onko python.exe-ohjelman sisältävä hakemisto lisätty käyttöjärjestelmän PATH-muuttujaan. Pythonin asennuksesta puhuttiin luvussa 8, joten sieltä löytyy tarkempia ohjeita tähän liittyen. Komentorivistä on useita erilaisia toteutuksia ja Windows, Apple ja Linux -komentorivit näyttävät ja toimivat hieman eri tavoin. Windows'ssa on DOS-perintönä tulleen cmd.exe -komentorivin lisäksi tarjolla PowerShell, jossa on myös omat erityispiirteet. Käytännössä uuden komentoriviympäristön opetteluun pitää siis käyttää oma aikansa, jos sitä rupeaa käyttämään enemmän.

Yhteenveto

Osaamistavoitteet

Ymmärrät graafisen käyttöliittymän perusteet ja tiedät miten edetä jatkossa niiden parissa, jos tarvetta ilmenee. Samoin komentoriviohjelma ja komentoriviparametrit ovat tuttuja niin käsitteinä kuin toiminta-ajatuksena ja pystyt perehtymään niihin tarkemmin tarpeen mukaan.

Loppusanat

Olemme tämän oppaan 13 luvussa tutustuneet ohjelmoinnin perusteisiin Python-kielen ja ohjelmistokehityksen perusajatusten kanssa. Tähän pisteeseen päästyämme voimme jo alkaa puhua ohjelmoinnista laajemmin sekä ohjelmistotuotannosta tavoitteena tehdä oikeita ohjelmia sen sijaan, että rakentelemme esimerkkejä olemassa olevien rakenteiden päälle. Tästä eteenpäin loogisia jatkoaiheita ovat mm. olio-ohjelmointi ja graafiset käyttöliittymät sekä toisaalta laitteistoläheisemmät ohjelmointikielet, testaus ja tietokannat.

Python-ohjelmointiin liittyviä materiaalia löytyy paljon Internetistä etenkin englannin kielellä. Verkon suurinta linkkikirjastoa Python-materiaaliin ylläpitää PSF:n wiki-kirjasto osoitteessa http://wiki.python.org/moin/ ja Pythonin omiin dokumentteihin kannattaa tutustua osoitteessa http://docs.python.org/py3k. Pythonin suomenkielisiä materiaaleja kannattaa katsoa Python Software Foundationin verkkokirjastosta osoitteessa http://wiki.python.org/moin/FinnishLanguage, vaikka kaikki lähteet eivät tällä hetkellä näytä olevan ajan tasalla. Oppaassa käsiteltiin muuttujan rooleja, joista lisää luettavaa löytyy Jorma Sajaniemen sivuilta osoitteessa http://saja.kapsi.fi/var_roles/.

Python on siitä hyvä ohjelmointikieli, että voit halutessasi jatkaa vielä kauan kielen kanssa ohjelmointia. Tässä oppaassa käsiteltyjen perusasioiden ymmärtäminen on merkki siitä, että pystyt halutessasi tekemään paljon muitakin asioita kuin mitä tässä oppaassa käsiteltiin. Vaikka et tietäisi tarvitsevasi ohjelmointitaitoja jatkossa, sinun ei kannata täysin unohtaa nyt oppimiasi asioita: nykytietotekniikalla ohjelmointi on aihe, joka tulee vastaan hyvinkin yllättävissä paikoissa, kuten Excel-taulukoiden tai interaktiivisten PowerPoint-esitysten yhteydessä. Lisäksi, mikäli olet tietotekniikan opiskelija, on sinun hyvä opetella Pythonia, koska se on edelleen vahvassa kasvussa oleva kieli, jonka käyttäjäkunta kasvaa koko ajan.