PYTHON

-

TKINTER JA GRAAFINEN KÄYTTÖLIITTYMÄ

Lappeenrannan teknillinen yliopisto 2007
Jussi Kasurinen
ISBN 978-952-214-401-0 ISSN 1459-3092

9

PYTHON – TKINTER JA GRAAFINEN KÄYTTÖLIITTYMÄ

Jussi Kasurinen

Käsikirjat 9
Manuals 9

PYTHON – TKINTER JA GRAAFINEN KÄYTTÖLIITTYMÄ

Jussi Kasurinen

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan osasto
PL 20
53851 Lappeenranta

ISBN 978-952-214-401-0
ISSN 1459-3092

Lappeenranta 2007

Materiaali, esimerkit sekä taitto:

Jussi Kasurinen

Oikoluku, kommentointi:

Uolevi Nikula

Lappeenrannan teknillinen yliopisto, Tietojenkäsittelytekniikan laboratorio.

Lappeenranta 05.06.2007

ESIPUHE

Tämä opas on kolmas Lappeenrannan teknillisen yliopiston tietojenkäsittelytekniikan laboratorion opas, ja se on tarkoitettu perehdyttämään lukija Tkinter-käyttöliittymäkirjaston alkeisiin. Oppaan tarkoituksena ei ole olla täydellinen käsikirja Tkinter-käyttöliittymäohjelmointiin, vaan ensisijaisesti antaa valmiuksia ymmärtää Tkinter-koodin rakennetta sekä toteuttaa itse yksinkertaisia graafisia käyttöliittymiä.

Oppaassa olevat esimerkkikoodit on tehty ja testattu Windows XP-työasemalla käyttäen Python-tulkin versiota 2.5. Lisäksi oppaan esimerkeissä ja ennakkotiedoissa oletetaan, että käyttäjällä on Python-ohjelmointikielen perustiedot mm. lähdekooditiedostojen käyttämisestä ja suorittamisesta, kielen perusrakenteista sekä tietorakenteista.

Mikäli Python-ohjelmointikieli on sinulle uusi, niin on suositeltavampaa, että aloitat opiskelun aiemmin julkaistusta Python-ohjelmointioppaasta [1] (Tietojenkäsittelytekniikan käsikirjat 7, LTY 2006). Opas on ilmainen ja aloittaa aivan kielen alkeista. Se on suositeltava tukiopas tämän kirjan aiheisiin tutustuttaessa, sillä useimmissa esimerkeissä käydään tarkemmin läpi ainoastaan käyttöliittymän toteuttamiseen liittyviä komentoja; mikäli toteutettavan ohjelman rakenne on monimutkainen, saatetaan myös peruskomentoja kuvata lyhyesti.

Tkinter-kirjastosta on myös oma englanninkielinen käsikirja [2], joka on kehitystiimin toimittama. Tämä käsikirja on enemmän tietosanakirjamainen ja suunnattu ennemminkin kokeneiden ohjelmoijien käytettäväksi. Kyseiseen teokseen on sisällytetty lyhyt tutoriaali, joka kuitenkin on lähinnä pintapuolinen silmäys kirjaston toimintalogiikkaan. Lisäksi verkosta on saatavilla myös muita ohjeita, mutta pääosin englannin kielellä.

SISÄLLYSLUETTELO

Valmistelut

Ennakkovalmistelut Tkinter-moduulia varten

Python-ohjelmointikielessä voidaan toteuttaa graafisia käyttöliittymiä Tkinter-kirjastolla, joka tulee automaattisesti Python-ohjelmointiympäristön mukana eikä sitä varten tarvita erillisiä asennustoimenpiteitä.

- Luonnollisesti tämä tarkoittaa sitä, että voidaksesi aloittaa Tkinter-moduulin kanssa työskentelyn, on sinun ensin asennettava työasemallesi Python-tulkki. Esimerkiksi Python Software Foundationin kotisivuilta [3] löydät asennuspaketin, jonka koko on käyttöjärjestelmästä riippuen vajaasta kymmenestä kahteenkymmeneen megatavua. Huomioi, että mikäli työasemassasi on 64-bittinen suoritin, voit ladata myös sitä varten optimoidun version.

- Lisäksi huomioi, että Python-tulkista on olemassa kaksi vastikään julkaistua versiota; 2.5.1 ja 2.4.4. Tämän oppaan esimerkit ja mallikoodit on testattu versiolla 2.5.1, mutta niiden pitäisi toimia ongelmitta myös version 2.4.4 tulkilla. Tarkemmat erot versioiden välillä löydät niiden julkaisudokumentaatioista, jotka nekin löytyvät Python Software Foundationin sivuilta.

- Tarkemmat asennusohjeet ohjelmointiympäristölle löydät perusohjelmointioppaasta [1]. Oppaan versiossa 1 puhutaan vielä Python-tulkin versiosta 2.4.3, mutta muutamia graafisia muutoksia lukuun ottamatta itse asennusvaiheet ovat pysyneet samoina.

Luku 1: Ensimmäinen Tkinter-ohjelma

Perusteet

Nykyisin useimmat ohjelmat julkaistaan käyttöjärjestelmille, joissa on mahdollisuus käyttää graafista käyttöliittymää. Toisin kuin vielä esimerkiksi 15 vuotta sitten, 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 hiiren avulla valita mitä haluaa tehdä. Monesti aloitteleva ohjelmoija kuitenkin luulee, että tällaisen käyttöliittymän toteuttamista varten tarvitaan jokin monimutkainen tai kallis kehitystyökalu tai että se olisi erityisen vaikeaa. Ehkä joidenkin ohjelmointikielien yhteydessä tämä voi pitää paikkansa, mutta Pythonissa yksinkertaisten graafisten käyttöliittymien tekeminen onnistuu helposti Tkinter-ohjelmamoduulin 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 pysyy edelleen samana. Koodi voidaan edelleen tuottaa yksinkertaisesti tekstieditorilla, josta tulkki tuottaa graafisen esityksen. Kannattaa kuitenkin huomata, että Tkinter-ohjelmakoodi on käytännössä aina hyvin pitkä ja monesti myös melko työläs kirjoitettava, joten koodin ajamista suoraan tulkin ikkunassa ei edes kannata yrittää. Joka kerta kun käytämme Tkinter-moduulia, on käytännössä ainoa järkevä lähestymistapa ongelmaan luoda lähdekooditiedosto, joka tallennetaan ja ajetaan sellaisenaan tulkista.

Seuraavissa viidessä luvussa tutustumme siihen, kuinka yksinkertaisten käyttöliittymien ja sen osien, kuten tekstikenttien, painikkeiden, valintanappien tai alasvetovalikoiden tekeminen onnistuu. Tämä tietenkin tarkoittaa, että ensimmäiseksi tarvitsemme ikkunan, johon käyttöliittymän rakennamme.

Graafinen käyttöliittymä

Esimerkki 1.1: Perusikkuna

Esimerkkikoodi

```
# -*- coding: cp1252 -*-

from Tkinter import *

pohja = Tk()
pohja.mainloop()
```

Esimerkkikoodin tuottama tulos

Ajamme esimerkin koodin, tuottaa tulkki seuraavanlaisen käyttöliittymäikkunan:

Kuva 1: Esimerkin 1.1 tuottama käyttöliittymäikkuna.

Kuinka koodi toimii

Ensimmäinen esimerkkikoodi on varsin lyhyt, eikä vielä sisällä paljoakaan toimintoja. Ensimmäisellä rivillä meillä on aikaisemmistakin oppaista tuttu näppäimistökartan määrittely, jolla saamme käyttöömme skandinaaviset merkit sekä muut laajennetun ANSI-standardin mukaiset merkit. Toisella rivillä otamme käyttöön Tkinter-kirjaston. Tällä kertaa kannattaa huomata, että sisällytys toteutetaan hieman tavallisuudesta poikkeavalla from...import * -syntaksilla, johtuen siitä että sen käyttäminen on tämän kirjaston yhteydessä näppärämpää.

Kolmannella rivillä luomme käyttöliittymän perustan tekemällä muuttujasta `pohja` Tk()-luokan juuri- eli pohjatason (*root*). Tähän pohjatasoon alamme jatkossa lisäämään kaikki haluamamme

esineet ja toiminnot, tällä kertaa kuitenkin riittää että jätämme sen tyhjäksi. Neljännellä rivillä käynnistämme Tkinter-käyttöliittymän kutsumalla pohjatasoa sen käynnistysfunktiolla `mainloop`. Ohjelma lähtee käyntiin, ja tuottaa tyhjän ikkunan. Tämä tapahtuu sen vuoksi, että ohjelman pohjatasolle ei ole sijoitettu mitään muuta komponenttia. Jos olisimme laittaneet sille vaikka painonapin, olisi ruutuun ilmestynyt pelkkä painonappi.

Komponenttien lisääminen

Käyttöliittymästä ei ole kuitenkaan 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. `Frame`-komponentista puhumme lisää myöhemmin, tarkastelkaamme ensin kuinka pohjatasolle lisätään vaikkapa tekstikenttä.

Esimerkki 1.2: Tekstikenttä perusikkunassa

Esimerkkikoodi

```
# -*- coding: cp1252 -*-

from Tkinter import *

pohja = Tk()

tekstikentta = Label(pohja, text="Moi maailma!")
tekstikentta.pack()

pohja.mainloop()
```

Esimerkkikoodin tuottama tulos

Ajamme esimerkin koodin, tuottaa tulkki seuraavanlaisen käyttöliittymäikkunan:

Kuva 2: Esimerkin 1-2 ikkuna.

Kuinka koodi toimii

Tässä lähdekoodissa tuotamme ensimmäisen käyttöliittymän joka tekee jotain. Alkuun suoritamme samanlaiset alustustoimenpiteet kuten aiemmin. Määrittelemme näppäimistökartan, otamme käyttöön Tkinter-moduulin sekä luomme pohjatason muuttujaan `pohja`. Tästä eteenpäin määrittelemme ensimmäisen komponenttimme.

Tällä kertaa luomme komponentin nimeltä Label, jota käytetään tavallisimmin staattisen tekstin esittämiseen. Tämä sopiikin meille varsin hyvin, koska haluamme ainoastaan saada ohjelman tuottamaan tekstiä aiemmin tekemäämme ikkunaan. Tässä tapauksessa luomme komponentin muuttujaan `tekstikentta`, ja annamme sille määrittelyssä seuraavat parametrit:

1) Ensimmäinen parametri `pohja` tarkoittaa sitä, että luomme komponentin pohjatason päälle. Tämä tarkoittaa sitä, että käynnistäessämme ohjelman, ilmestyy luomamme tekstikenttä tekstikentta pohjatason määrittelemän alueen sisään. Tähän annetaan tavallisesti parametrina se ikkunan osa tai alue, johon napin halutaan ilmestyvän, mutta koska meillä ei ole käytössä olevaa ikkunointiasettelua niin komponentti voidaan laittaa suoraan pohjalle.
2) Toinen parametri määrittelee yksinkertaisesti sen, mitä tekstikentässä on tarkoitus lukea. Luonnollisesti parametriksi voi antaa myös muuttujannimen, mutta ilman lisätoimenpiteitä teksti luetaan muuttujasta ainoastaan alustuksen yhteydessä. Tekstikentän päivittämisestä puhutaan myöhemmin lisää.

Seuraavaksi paketoimme kyseisen komponentin. Tällä käskyllä ilmoitamme tulkille, että emme anna kyseiselle komponentille enempää alustusmääritteitä ja että komponentin voi ”pakata” käyttöliittymään. Pakkauksen yhteydessä voimme vielä antaa joitain sijoittelua koskevia käskyjä, mutta tällä kertaa niitä ei tarvita. Muista, että **ainoastaan pakattu komponentti näkyy käyttöliittymän ikkunassa**. Jos komponenttia ei muisteta pakata, ei tulkki koskaan piirrä sitä ruudulle ja silloin luultavasti käyttöliittymäsi toimii väärin. Lopuksi laitamme ohjelman käyntiin kutsumalla pohjatason `mainloop`-funktiota.

Huomautus komponenteista

Varmaan huomasit, että koodissa oli jo nyt useita parametreja sekä jäsenfunktioita. Useimmilla komponenteilla on monia sen ominaisuuksia kuten kokoa, väriä, fonttia, sijaintia ja muita esitykseen vaikuttavia asioita säätelevät parametrit, joita kaikkia ei tässä oppaassa ole järkevää käydä läpi. Oppaassa pyritään lähinnä siihen, että ymmärrät kuinka asiat Tkinter-moduulin avulla toimivat, jotta pääset alkuun käyttöliittymäohjelmoinnin kanssa. Tämän vuoksi onkin suotavaa, että oppaan esimerkkeihin tutustumisen lisäksi tutustut myös kehitysryhmän omaan referenssiaineistoon [2], josta löydät tarkat tiedot eri komponenttien muuttujista sekä niiden vaikutuksesta lopputulokseen.

Yhteenveto

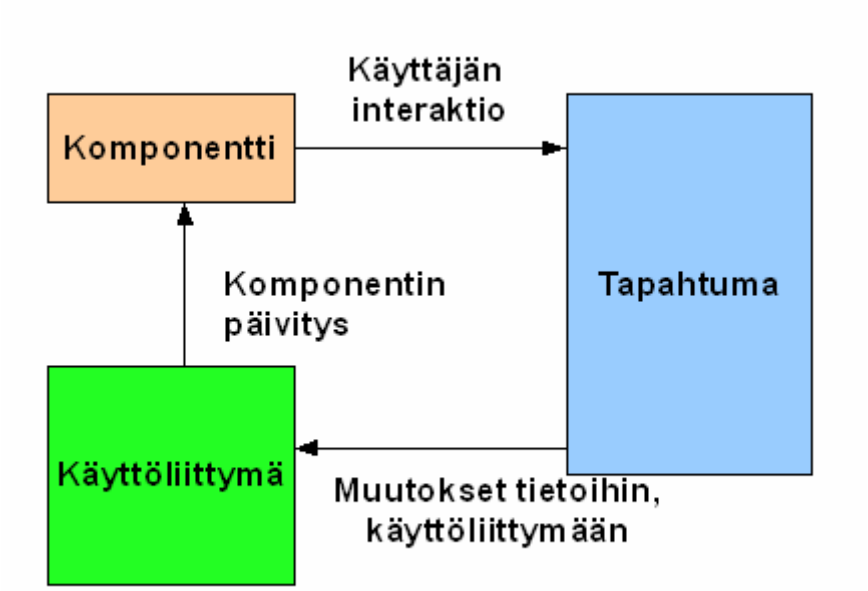
Tähän mennessä olemme tutustuneet Tkinter-kirjaston alustamiseen, sekä luoneet staattisen tekstiikkunan. Seuraavassa luvussa aloitamme tuottamaan ohjelmaan toiminnallisuutta sekä tarkastelemme hieman kuvien käyttämistä. Lisäksi esittelemme termin “tapahtumapohjainen ohjelmointi”, sekä selitämme miten se liittyy Tkinter-ohjelmointiin.

Luku 2: Interaktiivisuutta painikkeilla

Perusteet

Edellisessä luvussa loimme kaksi yksinkertaista graafista käyttöliittymää, joissa kuitenkaan ei ollut varsinaista toiminnallisuutta sen enempää kuin mitä käyttöjärjestelmä niille automaattisesti antoi. Toisessa luvussa tutustummekin Tkinter-moduulin tapaan toteuttaa toiminnallisuutta painonapin avulla.

Tkinteriin voidaan luoda komponenttien avulla esimerkiksi painonappeja, jotka ilmestyvät ruudulle normaalisti määrittelyn ja pakkauksen jälkeen. Kuinka sitten voimme päättää siitä, mitä nappi tekee? Yksinkertaisinta olisi määritellä toiminto joka suoritetaan aina kun nappia painetaan, eli luoda toimintaohjeet tilanteeseen, jossa käyttäjä painaa nappia. Juuri näin Tkinterin painonapille saadaan määriteltyä toiminnallisuuksia: luomme tapahtumia varten aliohjelman, joka liitetään komponenttiin. Aina, kun komponentille tehdään jotain, suoritetaan siihen liitetty aliohjelma. Tässä yhteydessä puhutaan usein tapahtumapohjaisesta ohjelmoinnista, jossa ohjelma odottaa käyttäjän interaktiota ja suorittaa tapahtumia sitä myötä kun käyttäjä niitä laukaisee.



Kuva 3: Tapahtumapohjainen ohjelmointi.

Jokaiselle komponentille, johon liittyy toiminnallisuutta, tulee määritellä aliohjelma tilanteisiin joissa käyttäjä tekee jotain komponentille. Suoritettava aliohjelma määritellään komponentin alustuksessa ja se käytännössä on normaalia ohjelmakoodia, joka ajetaan kun siihen liitettyä komponenttia käytetään. Tämä siis tarkoittaa sitä, että aliohjelma voi vapaasti suorittaa mitä tahansa operaatioita, kuten lukea tiedostoja, muuttaa käyttöliittymän tietoja, laskea tuloksia jne., eli käytännössä kaikkea mitä normaali ohjelma voi suorittaa. Kannattaa kuitenkin huomata, että tapahtuma ei palauta mitään arvoa komponenteille. Tämän vuoksi emme puhu *funktioista* vaan aliohjelmista, ja itse asiassa tavallisesti puhutaankin tapahtuman yhteydessä suoritettavasta aliohjelmasta tapahtumankäsittelijänä (*callback)*. Normaalisti ohjelmassa on ainakin yksi määritelty tapahtumankäsittelijä, eli lopetus. Tietenkään tätäkään ei tarvita, jos ohjelma on tarkoitettu lopetettavaksi käyttöjärjestelmän lopetuskäskyllä, eli ikkunan sulkemisella sitä varten varatusta rastista oikeassa yläkulmassa. Toinen tavallinen tapahtumakäsittelijä, varsinkin käyttöliittymän suunnitteluvaiheessa, on niin sanottu tyhjä käsittelijä, joka ei tee mitään mutta määrätään komponentille jotta tulkki suostuisi pakkaamaan sen. Seuraavaksi tutustumme tapahtumankäsittelijän määrittelemiseen käyttöliittymässä.

Graafinen käyttöliittymä

Esimerkki 2.1: Painikkeiden avulla valitseminen

Esimerkkikoodi

```
# -*- coding: cp1252 -*-

from Tkinter import *

def lopeta():
    import sys
    root.destroy()
    sys.exit(0)

root = Tk()
kehys = Frame(root)
kehys.pack()

tekstikentta = Label(kehys, text="Moi maailma!")
tekstikentta.pack()

button = Button(kehys, text="QUIT", command=lopeta)
button.pack(side=BOTTOM)

root.mainloop()
```

Esimerkkikoodin tuottama tulos

Kun suoritamme yllä olevan lähdekoodin Python-kääntäjällä, saamme tulokseksi seuraavanlaisen käyttöliittymäikkunan.

Kuva 4: Esimerkin 2.1 tuottama ikkuna.

Kuinka koodi toimii

Ohjelma alkaa ensin normaalilla aloituskommentilla jolla määritellään käytettävä näppäinkartta, sekä tämän alla olevalla Tkinter-moduulin käyttöönotolla. Heti tämän jälkeen törmäämmekin uuteen asiaan eli tapahtumankäsittelijän määrittelyyn.

Kun aiemmin mainitsimme, käytetään Python-ohjelmoinnissa tapahtumapohjaista lähestymistapaa käyttöliittymän komponenttien yhteydessä. Kun käyttäjä aiheuttaa tapahtuman, ajetaan tapahtuman kohteena olleen komponentin tapahtumankäsittelijä. Nämä käsittelijät tallennetaan ohjelmakoodiin eräänlaisina aliohjelmina, joissa oleva koodi ajetaan kun haluttu tapahtuma laukaistaan. Tässä tapauksessa olemme määritelleet käsittelijän nimeltään `lopeta`, jolla ohjelmassa ajetaan järjestelmäkäsky `sys.exit(0)`, joka lopettaa ohjelman sekä `root.destroy()`, joka tuhoaa käyttöliittymäikkunan ja vapauttaa sen varaaman muistialueen. Tässä tärkeää on huomata tapahtuman nimi – `lopeta` - joka myöhemmin koodissa annetaan komponentille sen laukaiseman tapahtumankäsittelijän nimeksi.

Tapahtumankäsittelijän määrittelyn jälkeen palaamme lähdekoodiin, jossa alustamme pohjatason muuttujaan `root`. Tämän jälkeen teemme toisen uuden toimenpiteen, alustamme pohjatason päälle kehyskomponentin `Frame`. Tämä liittyy siihen, että kehyksien avulla voimme tarkemmin ohjata komponenttien järjestystä käyttöliittymässä määrittelemällä sisäkkäisten kehysten avulla pohjatason päälle kehikon, johon komponentit sijoitellaan. Tästä puhumme myöhemmin lisää, nyt riittää kun ymmärrät että luomme pohjatason päälle `kehys`-nimisen komponentin, jonka sisään sijoitamme kaikki loput komponenttimme. Kehys pakataan kuten muutkin komponentit, jotta saamme sen sisältämät osat myöhemmin näkyviin.

Seuraavilla kahdella rivillä määrittelemme tekstiruudun, johon asetamme tekstiksi ”Moi maailma”:n. Huomaa, että tällä kertaa sijoitimme tekstiruudun kehyskomponentin `Frame` sisään, emmekä suoraan pohjatasolle. Tämän lisäksi teemme toisen komponentin nimeltä `Button`, jolla saamme käyttöliittymäämme painonapin. `Button` ottaa vastaan kolme parametria. Ensimmäisellä parametrilla määrätään, mihin kehykseen nappi sijoitetaan, toisella ilmoitetaan, mikä teksti nappiin laitetaan ja kolmannella määrätään, mikä tapahtuma suoritetaan kun nappia painetaan. Tällä kertaa

emme ohjelmoineet käyttöliittymäämme muita tapahtumia kuin tapahtuman `lopeta`, joten annamme napille sen tapahtuman. Komponentti `Button`ille voidaan antaa myös muita arvoja, kuten napin väri tai koko, mutta näistä voit lukea enemmän esimerkiksi kehitystiimin omasta käyttöoppaasta [2].

Tässä vaiheessa kannattaa huomauttaa siitä, että pystyäksesi suorittamaan koodin täytyy jokaiselle tapahtuman vaativalle komponentille antaa sellainen. Tämä on harmillista, jos haluamme ainoastaan testata käyttöliittymän ulkonäköä ilman että haluamme koodata työhön varsinaista toiminnallisuutta. Tässä tapauksessa onkin tavallista käyttää ns. tyhjää käsittelijää täyttämään annettu tehtävä, eli luoda käsittelijä joka ei tee mitään. Yksinkertaisimmillaan tämä voidaan toteuttaa koodilla

```
def tyhja():
    pass
```

Tässä tapauksessa tapahtumakäsittelijän `tyhja` ajaminen ei tee mitään, mutta tulkki on tyytyväinen koska komponentille on määritelty käsittelijä. Tapahtuman määrittelyn jälkeisellä rivillä luomamme nappi lisäksi pakataan. Koska haluamme, että käyttämänne painonappi on tekstikentän alapuolella, määritellään tämä vielä erikseen pakkausvaiheessa parametrilla `side`, jonka arvoksi annetaan BOTTOM. Parametri `side` määrää sen, mille puolelle edellisiä samaan kehykseen sijoitettuja komponentteja uusi komponentti sijoitetaan. Jos olisimme antaneet arvoksi TOP, olisi painike tullut tekstikentän yläpuolelle. Vastaavasti arvoilla LEFT tai RIGHT sijoitus olisi ollut vastaavasti vasemmalla tai oikealla puolella. Oletusarvona Tkinter käyttää arvoa BOTTOM.

Entäpä jos haluaisimme tavallisen tekstikentän sijaan käyttää ruudulla kuvaa? Esimerkiksi ladata tekstiruudun tilalle kuvan tai muita graafisia elementtejä? Tällöin käytämme komponenttia nimeltä `Canvas`.

Esimerkki 2.2: Kuvan esittäminen käyttöliittymäikkunassa

Esimerkkikoodi

```
# -*- coding: cp1252 -*-

from Tkinter import *

def lopeta():
    import sys
    root.destoy()
    sys.exit(0)


root = Tk()
root.title("Esimerkkiohjelma")
frame = Frame(root)
frame.pack()

kanvas = Canvas(frame, width = 400, height = 240)
kuva = PhotoImage(file = "mallikuva_1.gif")
kanvas.create_image(200,120,image = kuva)
kanvas.pack(side = TOP)
```

```
button = Button(frame, text="QUIT", command=lopeta)
button.pack(side=BOTTOM)

root.mainloop()
```

Esimerkkikoodin tuottama tulos

Kun ajamme yllä olevan koodin, saamme tulokseksi seuraavanlaisen käyttöliittymäikkunan:

Kuva 5: Esimerkin 2.2 tuottama käyttöliittymäikkuna.

Kuinka koodi toimii

Luvun toisessa esimerkkikoodissa on kaksi varsinaista muutosta edelliseen koodiin verrattuna. Ensinnäkin olemme antaneet ohjelmallemme ensi kertaa nimen ikkunan otsikkotietoihin. Tämä voidaan määritellä pohjatasolle annettavan jäsenmuuttujan `title` arvona joka on ensi kertaa määritelty omalla arvollaan. Annoimme ohjelman nimeksi ”Esimerkkiohjelma” joka ilmestyi aukeavan ikkunan otsikoksi.

Toinen muutos koodiin on Label-tekstikentän korvaaminen Canvas-komponentilla. Canvas on Tkinter-moduulin yleiskomponentti, jonka avulla voidaan esittää grafiikkaa, tekstiä sekä muistista ladattuja kuvia. Koska haluamme nimenomaisesti esittää kuvan, ovat toimenpiteet seuraavanlaiset: ensin luomme Canvas-komponentista pohjan `kanvas` sekä lataamme kuvatiedoston `mallikuva_1.png` funktiolla `PhotoImage` kuvamuuttujaan `kuva`. Kun olemme tehneet molemmat esivalmisteluvaiheet, voimme ladata kuvan kanvakselle funktiolla `create_image`. Tämä funktio ottaa parametreina vastaan kaksi interger-arvoa, joilla kuva asetetaan x- ja y-suunnassa komponenttiin. Lisäksi funktiolle annetaan luotu kuvamuuttuja, josta kuva ladataan. Lopuksi komponentti pakataan, jotta se tulisi näkyviin.

Yhteenveto

Tässä luvussa esittelimme lyhyesti Tkinter-järjestelmän käyttämän tapahtumapohjaisen lähestymistavan käyttöliittymän toimintaan. Lisäksi loimme ensimmäisen oikean ohjelmamme, jossa oli tapahtumapohjaista interaktiota. Tutustuimme myös komponenteista painonappiin sekä kanvakseen. Seuraavassa luvussa jatkamme erilaisiin komponentteihin tutustumista ja tutkimme tapoja, joilla voimme lähettää käyttöliittymällemme vapaamuotoista tekstiä.

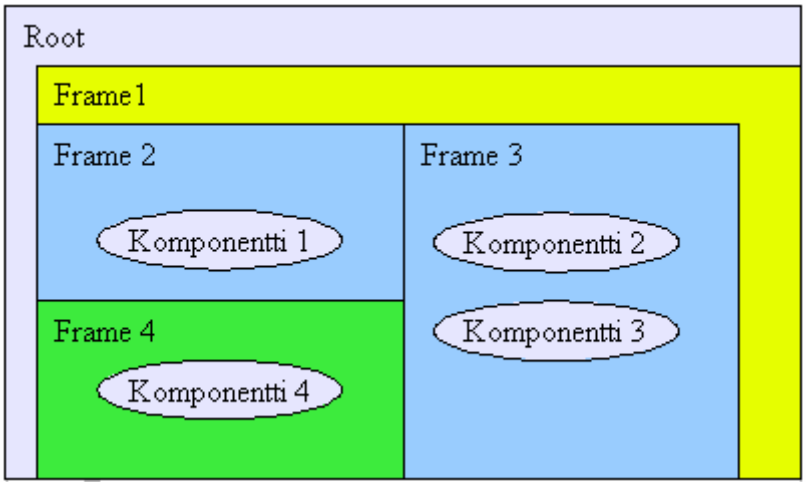
Luku 3: Syötteiden antaminen

Perusteet

Tässä luvussa käsittelemme syötteiden antamista sekä komponenttien järjestelyä käyttöliittymäikkunassa. Edellisessä luvussa jo sivusimme komponenttien järjestelyä puhumalla komponenttien asettelusta toisiinsa nähden sekä alustimme aihetta luomalla pohjatason päälle yksinkertaisen Frame-komponentin.

Ikkunoinnin asettelusta

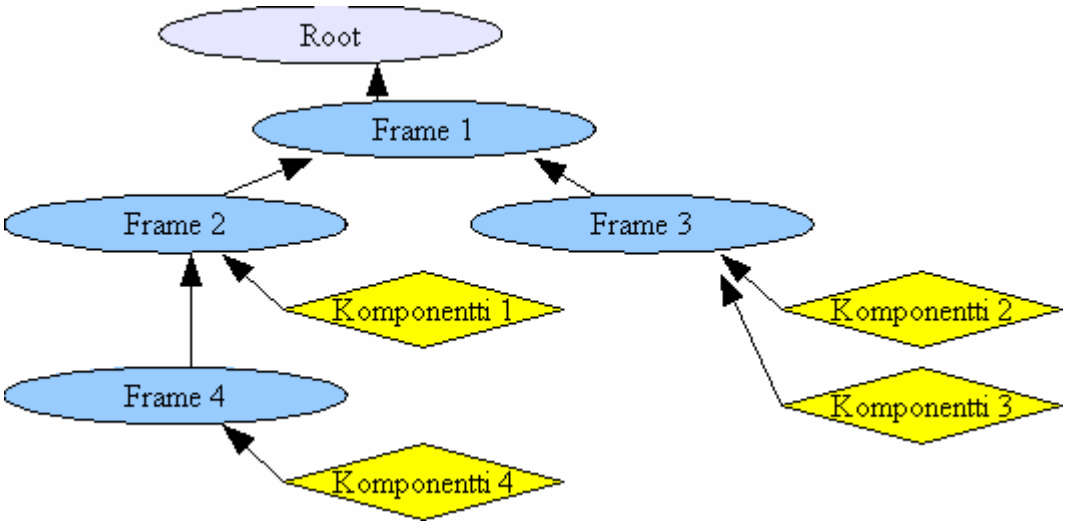
Ikkunoinnin asettelu aloitetaan aina Frame-komponentin lisäämisellä pohjatason päälle. Tällöin saamme perustan, jonka päälle voimme sijoittaa muita komponentteja tai asettelua varten varattuja muotoja. Koska Tkinter-kirjastossa on hyvin rajallisesti vaihtoehtoja komponenttien keskinäiselle sijoittelulle – käytännössä TOP, BOTTOM, LEFT ja RIGHT, jotka nekin vaikuttavat vain kahden komponentin väliseen sijoitteluun – joudumme usein tilanteeseen, jossa haluamme määritellä tarkemmin, mikä komponentti menee minnekin. Tällöin voimme sijoittaa Frame-komponentteja sisäkkäin ja muodostaa niiden avulla halutun kaltaisen muodon. Tämän mahdollistaa Tkinter-kirjaston ominaisuus, joka poistaa turhan tilan komponenttien ympäriltä. Tämä näkyy ulospäin esimerkiksi käyttöliittymäikkunan koon muuttumisena tekstin pituuden mukaisesti, kuten ensimmäisen luvun esimerkeistä huomasimme. Kun asetimme ikkunaan tekstikentän, pieneni oletusikkuna automaattisesti juuri tekstikentän kokoiseksi. Käyttöliittymän suunnittelun yhteydessä tätä voidaan käyttää myös hyödyksi. Jos taas komponentilla ei ole ulospäin näkyviä osia, ei komponenttia esitetä käyttöliittymäikkunassa. Tämän vuoksi voimme vapaasti sijoitella Frame-komponentteja sisäkkäin muotoilun toteuttamiseksi, koska tämä ei kuitenkaan näy ulospäin loppukäyttäjälle. Joudumme ainoastaan huolehtimaan siitä että emme määritä Frame-komponenteille yhtäkään näkyvää osaa, kuten esimerkiksi reunaviivoja tai vastaavaa.



Kuva 6: Ikkunoinin suunnittelu Tkinter-komponenteilla.

Jos haluaisimme esimerkiksi kuvan 6 kaltaisen pohjaikkunoinnin, joudumme aloittamaan siten, että luomme pohjatason päälle Frame-komponentin. Kutsumme tätä esimerkissä nimellä `Frame 1`, ja sen tunnusvärinä on keltainen. `Frame 1`:n sisälle sijoitamme kaksi uutta Frame-komponenttia, `Frame 2`:n ja sen oikealle puolelle `Frame 3`:n. Näitä kuvaamme sinisellä pohjavärillä. Lopuksi vielä sijoitamme `Frame 2`:n sisään `Frame 4`:n, joka sijoitetaan pohjalle asettamalla align-muuttujalle arvo BOTTOM. Tämän jälkeen lisäämme vielä `Frame 2`:een, `Frame 3`:een ja `Frame 4`:ään muutaman komponentin kuten esimerkiksi painonapin. Yhdessä nämä komponentit sijoitetaan siten, että ne muodostavat kuvan 7 kaltaisen hierarkisen rakenteen. Tämä tarkoittaa sitä, että komponentti sijoitetaan nuolella kuvatun komponentin sisään. Tällöin komponentit voidaan sijoittaa yksinkertaisilla määritteillä (TOP, BOTTOM jne.) ja silti huolehtia siitä, että komponenttien suhteellinen sijainti pysyy aina oikeana.

Esimerkiksi komponentti 1 ei voi koskaan olla komponenttien 2 tai 3 oikealla puolella koska sen ”omistava” Frame 2 on komponenttien 2 ja 3 omistavan Framen vasemmalla puolella. Samoin komponentti 4 ei koskaan voi olla komponentin 1 yläpuolella, koska Frame 4 on määritelty aina olemaan Frame 2:n omien komponenttien alapuolella. Kannattaa kuitenkin huomata, että mikäli Frame 2:lle ei anneta omia komponentteja, nousee Frame 4 samalle tasolle kuin Frame 3. Tämä johtuu nimenomaan siitä, että Tkinter piilottaa Frame 2:n koska siinä ei ole mitään esitettävää eikä täten myöskään varaa sille tilaa käyttöliittymäikkunasta.



Kuva 7: Frame-komponenttien ja toiminnallisten komponenttien muodostama hierarkia.

Käyttöliittymäikkunan suunnittelu voi alkuun tuntua vaikealta, mutta ajattele asiaa siten, että jaat ikkunasi pysty- tai vaakasuorassa kahtia sijoittamalla Frame-komponentteja toistensa sisään BOTTOM- tai LEFT-parametreilla niin kauan kunnes olet saanut tarpeeksi monta ”osaikkunaa”. Lisäksi voit myös kokeilla Grid-komponentilla toteutettua ikkunanasettelua, mutta se ei kuulu tämän oppaan aihepiiriin. Grid-ikkuna-asettelusta voit lukea enemmän tekijätiimin käsikirjasta [2].

Syötteiden vastaanottaminen

Tämän luvun varsinainen aihe on syötteiden ottaminen käyttäjältä. Tällä tarkoitetaan siis tekniikkaa, jolla käyttäjä voi antaa graafisella käyttöliittymällä toimivalle ohjelmalle syötteenä merkkijonoja, tekstiä, numeroita tai mitä tahansa muuta mitä katsoo tarpeelliseksi. Tkinter-moduulissa tätä varten on olemassa oma komponentti, Entry. Tutustumme seuraavissa kahdessa esimerkissä kyseisen komponentin toimintaan.

Esimerkki 3.1: Syöteikkuna

Esimerkkikoodi

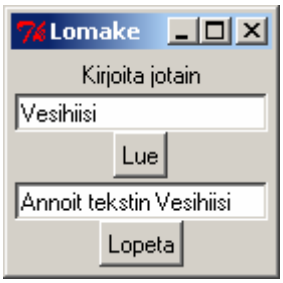
```
# -*- coding: cp1252 -*-

from Tkinter import *
import sys

def lue():
  sana = ""
  sana = str(Entry.get(syote))
  tulos = "Annoit tekstin "+sana
  Entry.delete(tuloste, 0, END)
  Entry.insert(tuloste, 0, tulos)

def lopeta():
    root.destroy()
    sys.exit(0)

root=Tk()
root.title("Lomake")
frame=Frame(root, borderwidth = 1)
frame.pack()
frame2 = Frame(frame, borderwidth = 1)
frame2.pack(side = TOP)
frame3 = Frame(frame, borderwidth = 1)
frame3.pack(side = BOTTOM)

ohje = Label(frame2, text = "Kirjoita jotain")
ohje.pack(side = TOP)

lue_nappi=Button(frame2, text="Lue", command=lue)
lue_nappi.pack(side=BOTTOM)
lopeta_nappi=Button(frame3, text="Lopeta", command=lopeta)
lopeta_nappi.pack(side=BOTTOM)

syote=Entry(frame2)
syote.pack(side=LEFT)
tuloste=Entry(frame3)
tuloste.pack(side=LEFT)


root.mainloop()
```

Esimerkkikoodin tuottama tulos

Kun suoritamme esimerkin 3.1 lähdekoodin, saamme seuraavanlaisen käyttöliittymäikkunan:

Kuva 8: Esimerkin 3.1 käyttöliittymä.

Kun kirjoitamme ylempään syötekenttään tekstin “Vesihiisi” ja painamme painiketta “Lue”, tulostuu alenmpaan syötekenttään teksti “Annoit tekstin Vesihiisi”. Ohjelma loppuu painikkeesta “Lopeta”.



Kuva 9: Esimerkin 3-1 käyttöliittymä käytössä.

Kuinka koodi toimii

Ohjelma aloitetaan normaaliin tapaan ensin määrittelemällä näppäimistökartta ja tämän jälkeen käyttöönottamalla Tkinter-kirjasto. Tällä kertaa tiedämme myös tekevämme ohjelmaan lopetustapahtuman, joten käyttöönotamme myös `sys`-kirjaston. Palaamme toteutettuihin tapahtumiin hetken päästä tarkemmin, tutkittuamme ensin itse pääohjelman rakennetta.

Tällä kertaa ohjelma hyödyntää Frame-komponenttien avulla toteutettua käyttöliittymän määrittelyä siten, että pohjatason päälle on määritelty yksi Frame-komponentti, jonka päällä olemme määritelleet kaksi allekkaista Frame:a. Ylempään Frame-komponenttiin, `Frame2`:een, olemme sisällyttäneet komponentteina tekstikentän `ohje`, syötekentän `syote` sekä painikkeen `lue_nappi`. Alempaan Frame-komponenttiin olemme sijoittaneet syotekentän `tuloste` ja painikkeen `lopeta_nappi`. Koodissa komponenttien määrittely on toteutettu siten, että ensin määrittelemme ikkunoinnin Frame-komponentit ja tämän jälkeen jokaisen toiminnallisen komponentin omina ryhminään; ensin tekstikenttä, sitten painikkeet ja viimeisenä syötekentät.

Tarkastelkaamme Entry-komponenttia hieman tarkemmin. Kuten huomaat, määrittelyvaiheessa Entry-kentälle ei tarvitse antaa kuin Frame-komponentti, johon se sijoitetaan. Tämän tehtävän

syötekenttien leveys määräytyy oletusarvoisesti kuvan mukaiseksi. Entry-komponentin varsinaiseen toimintaan pääsemme tutustumaan paremmin kutsusarjasta `lue`.

Ensinnäkin tekstikentän tiedot luetaan lukea komennolla `sana = Entry.get(syote)`. Tällä komennolla luemme Entry-komponentin `syote` arvon muuttujaan `sana`. Itse koodissa käskyn ympärillä on vielä tyyppimuunnosfunktio `str`, jolla varmistetaan että luettua arvoa käsitellään nimenomaan merkkijonona.

Vastaavasti syötekentän tyhjentämiseen tai sisältämän merkkijonon määräämiseen käytetään funktioita `delete` ja `insert`. Molemmat funktiot toimivat samankaltaisella syntaksilla, eli ensin määrätään mitä syötekenttää halutaan muuttaa ja tämän jälkeen määrätään parametreina miten muutos toteutetaan. Funktion `delete` tapauksessa parametrit ovat aloituspiste sekä lopetuspiste. Nämä tarkoittavat syötekenttään annettua merkkijonoa ja siinä olevien merkkien sijaintia. Tässä tapauksessa `0` tarkoittaa arvona paikkaa ensimmäisen syötetyn merkin edessä ja `END` sijaintia viimeisen merkin takana. Tämä tarkoittaa siis sitä, että käsky `Entry.delete(tuloste, 0, END)` poistaa syötekentästä `tuloste` kaikki merkit (ensimmäisen edestä viimeisen taakse). Myös muut numeroarvot toimivat tässä tapauksessa samalla logiikalla kuin esimerkiksi leikkauksissa. Leikkauksista voit lukea lisää perusohjelmointioppaan [1] luvusta 3.

Funktio `insert` toimii muuten samalla tavoin, mutta se saa parametrinsa seuraavasti: Toinen parametri on kohta, josta eteenpäin merkkejä aloitetaan lisäämään, ja kolmas parametri muuttujan arvo, joka lisätään syötekenttään. Huomioi kuitenkin, että `insert` ei tuhoa mitään aiempia merkintöjä syötekentästä. Tämän vuoksi varmistuaksesi siitä, että ohjelmaasi ei tule virheellisiä tulosteita, joudut tyhjentämään syötekentät käsin ennen uuden merkkijonon syöttämistä syötekenttään. Vaikka tässä esimerkissä pääasiassa toimittiinkin merkkijonojen parissa, ei numeroarvoilla työskentely poikkea merkittävästi tämän esimerkin koodista. Seuraavaksi tutustummekin yksinkertaiseen laskin-esimerkkiin, jossa harjoittelemme numeroarvojen käsittelyä syötekentillä.

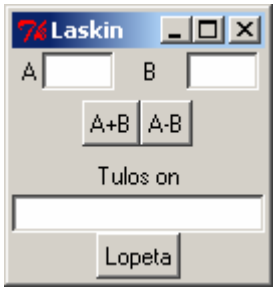
Esimerkki 3.2 Plus-minus-laskin

Esimerkkikoodi

```
# -*- coding: cp1252 -*-
from Tkinter import *
import sys

def plus():
  luku1 = float(Entry.get(arvo1))
  luku2 = float(Entry.get(arvo2))
  tulos = luku1 + luku2
  Entry.delete(vast, 0, 20)
  Entry.insert(vast, 0, tulos)

def minus():
  luku1 = float(Entry.get(arvo1))
  luku2 = float(Entry.get(arvo2))
  tulos = luku1 – luku2
  Entry.delete(vast, 0, 20)
  Entry.insert(vast, 0, tulos)

def lopeta():
    root.destroy()
    sys.exit(0)

root=Tk()
root.title("Laskin")
frame=Frame(root)
frame.pack()
frame2=Frame(root, borderwidth=0)
frame2.pack()
frame3=Frame(root, borderwidth=5)
frame3.pack()
frame4=Frame(root)
frame4.pack()

button1=Button(frame3, text="A+B", command=plus)
button1.pack(side=LEFT)
button2=Button(frame3, text="A-B", command=minus)
button2.pack(side=LEFT)
button6=Button(frame4, text="Lopeta", command=lopeta)
button6.pack(side=BOTTOM)

teksti1=Label(frame2, text="A")
teksti1.pack(side=LEFT)
arvo1=Entry(frame2, width=5)
arvo1.pack(side=LEFT)

teksti2=Label(frame2, text="B")
teksti2.pack(side=LEFT)
arvo2=Entry(frame2, width=5)
arvo2.pack(side=LEFT)

teksti3=Label(frame4, text="Tulos on")
teksti3.pack()
vast=Entry(frame4)
vast.pack(side=LEFT)


root.mainloop()
```

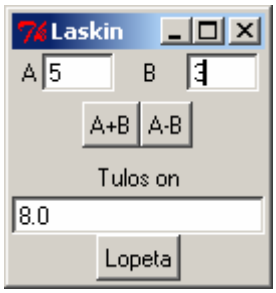
Esimerkkikoodin tuottama tulos

Kun ajamme yllä olevan lähdekoodin, saamme seuraavanlaisen käyttöliittymäikkunan:



Kuva 10: Esimerkin 3.2 käyttöliittymäikkuna.

Kun syötämme käyttöliittymän kenttiin A ja B numeroarvot 5 ja 3, voimme painikkeilla “A+B” ja “A-B” suorittaa laskutoimituksia, joiden tulokset tulevat näkyviin alapalkin tekstikenttään.



Kuva 11: Esimerkin 3.2 laskin käytössä.

Kuinka koodi toimii

Itse ohjelmakoodi on niin samanlainen edellisen esimerkin kanssa, että tällä kertaa käymme läpi ainoastaan koodikohtaiset muutokset ja yksityiskohdat. Ensinnäkin Entry-komponenttien kokoa on tällä kertaa rajoitettu käyttämällä määrittelyn yhteydessä parametria `width`. Tämän parametrin avulla voimme määritellä kuinka monta kirjainmerkkiä leveä syötekenttä tulee olemaan. Oletusarvo 20 on tässä tapauksessa korvattu arvolla 5.

Lisäksi Frame-komponenttien väliin jäävää tyhjää tilaa on kasvatettu määrittelyyn annetulla paramterilla `borderwidth`, jonka avulla voimme määritellä Frame-komponentin reunan ja reunimmaisen komponentin väliin jäävän tyhjän alueen minimiraja. Tällä tavoin saamme käyttöliittymän ikkunasta ”ilmavamman”, jolloin itse käyttöliittymä näyttää usein sopusuhtaisemmalta ja vähemmän ”täyteen ahdetulta”. Lisäksi kannattaa huomata syötekenttien arvojen ottamisen yhteydessä oleva tyyppimuunnos `float` jolla varmistetaan, että kentästä luettua arvoa käsitellään liukulukuna (desimaalilukuna). Ohjelmassa ei kuitenkaan ole virheensieppausta päällä, joten merkkijonon antaminen kaataisi ohjelman. Tähän voisimme varautua try-except-rakenteella, joka esitellään esimerkiksi perusoppaan [1] luvussa 10. Muuten koodi toimii kuten aiemman esimerkin tekstiä lukeva esimerkkiohjelma.

Yhteenveto

Tässä luvussa tutustuimme yksinkertaiseen tapaan toteuttaa Tkinter-käyttöliittymällä arvojen syöttäminen graafiseen käyttöliittymään. Kuten huomasimme, on tekstin ja numeroarvojen syöttäminen ja lukeminen yksinkertaista käytettäessä Entry-komponenttia. Toisaalta Entry-komponentilla ei voi syöttää kuin yhden rivin kerrallaan, mutta tällä erää voimme tyytyä siihen. Voit lukea lisää äsken kokeillusta Entry-komponentista Lundhin verkko-oppaasta [2].

Luku 4: Valikoista ja valintatyökaluista

Valintojen toteuttamisesta

Edellisessä luvussa kokeilimme käyttöliittymän komponenttia, jolla pystyimme antamaan syötteitä ohjelmalle. Kuitenkaan Entry-komponentti ei vielä riitä kaikkiin tilanteisiin ja käyttöliittymämalleihin. Monesti voimme haluta tehdä lisävalintoja tai käyttää jonkinlaista valikkotyökalua sen sijaan, että valitsisimme asioita syötteillä tai painikkeilla.

Tkinter-kirjastossa valikoiden ja valintojen tekemiseen on omat komponenttinsa. Alasvetovalikon voimme toteuttaa Menu-komponentilla, jolla saamme muodostettua listamaisen rakenteen josta voimme valita toimintoja joita haluamme käyttää. Komponentin Checkbutton avulla voimme taas luoda käyttöliittymäikkunaan valintaruutuja, joilla voimme antaa vaikkapa valintamuuttujien arvoja sekä toteuttaa käyttöliittymän syötteissä tarvittavia valintakytkimiä.

Alasvetovalikko

Alasvetovalikko on valikkomuoto, johon käytännössä kaikki tietokoneiden kanssa työskennelleet ovat törmänneet. Tämä Windows-käyttöliittymän perustyökalu on tehokas valikkomekanismi, jolla voimme helposti lajitella ja sijoitella toisiinsa liittyviä valintoja keskenään kuitenkin siten, että valikko itse on poissa tieltä aina silloin kun sitä ei tarvita.

Alasvetovalikon toteuttaminen tehdään Tkinter-kirjastossa Menu-nimisellä komponentilla. Tämä komponentti poikkeaa aiemmin esitellyistä siten, että se voidaan liittää ainoastaan pohjatasoon tai toiseen olemassaolevaan Menu-komponenttiin, toisinkin kuin esimerkiksi Frame- tai Button-komponentit. Lisäksi alasvetovalikon toteuttaminen vaatii hieman ennakkovalmisteluja, mutta käytännössä itse valikon tekeminen on nopeaa ja yksinkertaista. Helpointa valikon toteuttamisen selittäminen on esimerkin avulla, joten seuraavaksi katsommekin, miten alasvetovalikon toteuttaminen käytännössä tapahtuu.

Esimerkki 4.1: Alasvetovalikko

Esimerkkikoodi

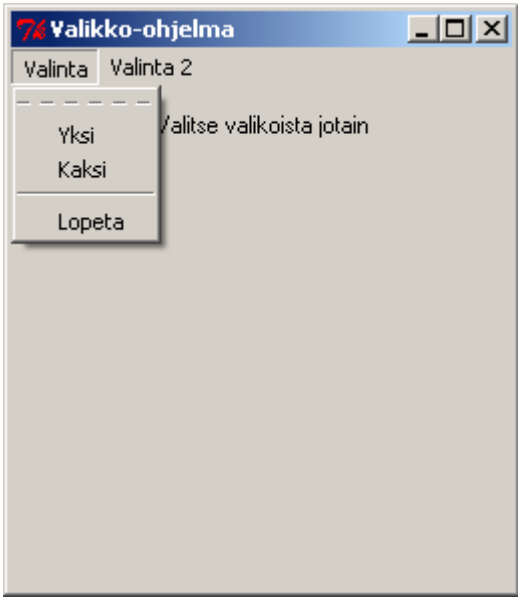
```
# -*- coding: cp1252 -*-
from Tkinter import *

def yksi():
    ruutu['text'] = "Valitsit yksi"
def kaksi():
    ruutu['text'] = "Valitsit kaksi"
def kolme():
    ruutu['text'] = "Valitsit kolme"
def nelja():
    ruutu['text'] = "Valitsit neljä"
def lopeta():
    root.destroy()
    sys.exit(0)

root = Tk()

valikko = Menu(root)
root.title("Valikko-ohjelma")
root.geometry("250x250")
root.config(menu=valikko)

tiedmenu = Menu(valikko)
valikko.add_cascade(label="Valinta", menu=tiedmenu)
tiedmenu.add_command(label="Yksi", command=yksi)
tiedmenu.add_command(label="Kaksi", command=kaksi)
tiedmenu.add_separator()
tiedmenu.add_command(label="Lopeta", command=lopeta)

muokmenu = Menu(valikko)
valikko.add_cascade(label="Valinta 2", menu = muokmenu)
muokmenu.add_command(label="Kolme", command=kolme)
muokmenu.add_command(label="Neljä", command=nelja)

frame= Frame(root)
frame.pack(side= TOP)
ruutu = Label(frame, padx = 10, pady = 10, text="Valitse valikoista jotain")
ruutu.pack()

mainloop()
```

Esimerkkikoodin tuottama tulos

Kun suoritamme yllä olevan esimerkin lähdekoodin, saamme seuraavanlaisen käyttöliittymäikkunan:

Kuva 12: Esimerkin 4.1 tuottama käyttöliittymäikkuna.

Valitsemalla hiirellä halutun valikon, voimme valita sieltä valinnan ”Yksi”, ”Kaksi”, ”Kolme” tai ”Neljä”. Ohjelma kertoo, minkä valinnan teit. Lisäksi voit valita myös valinnan ”Lopeta”, jolla ohjelma lopetetaan.



Kuva 13: Alasvetovalikko avattuna.

Kuinka koodi toimii

Koska koodissa on jälleen kerran paljon uutta asiaa, keskitymme ainoastaan uusiin asioihin. Ensinnäkin, kun tarkastelemme koodin rakenne huomaamme, että tällä kertaa olemme lisänneet pohjatasolle kaksi uutta määritelmää, `root.geometry("250x250")` ja `root.config( menu=menu)`.

Ensimmäinen komento, `root.geometry()` saa parametrinaan arvon ”250x250”, jolla määritellään kiinteästi se, minkä kokoinen käyttöliittymäikkunan tulee olla. Tämä siis tarkoittaa, että ikkuna ei enää muuta kokoaan ikkunassa olevien komponenttien mukaan, vaan säilyttää jatkuvasti vähintään koon 250 kertaa 250 pikseliä. Toisella komennolla, `root.config(menu = valikko)` määräämme, että aiemmin rivillä `valikko = Menu(root)` luotu `valikko`-niminen Menu-komponentti on päätason alasvetovalikko. Tämä tarkoittaa siis sitä, että käyttöliittymäikkunaan varataan otsikkopalkin ja varsinaisen ikkunan väliin tila, johon alasvetovalikko tullaan piirtämään.

Tässä vaiheessa meillä ei vielä ole varsinaista valikkoa, vaan ainoastaan määrittely sille, että käytämme alasvetovalikkoa ja komponentti, johon valikko on tarkoitus rakentaa. Itse alasvetovalikon toteuttaminen aloitetaan komennolla `tiedmenu = Menu(valikko)`. Tällä käskyllä luomme uuden Menu-komponentin `tiedmenu`, joka liitetään pohjatasoon kiinnitettyyn `valikko`-komponenttiin. Seuraavalla rivillä käyttämällä komentoa `valikko.add_cascade (label="Valinta",menu=tiedmenu)` saamme lisättyä alasvetovalikkoon ensimmäisen varsinaisen valikon nimeltään `Valinta`. Funktiolle `add_cascade` annammekin ensimmäisenä parametrina halutun ruudulla näkyvän valikon nimen ja toisena parametrina Menu-komponentin, jota tässä tapauksessa käytetään. Seuraavaksi luomme alasvetovalikkoon valintoja funktiolla `add_command`. Funktio `add_command` on Menu-komponentin jäsenfunktio, joten se tarvitsee parametreina ainoastaan ruudulla näkyvän nimen (`label`) sekä siihen kytketyn tapahtuman. Tämä siis tarkoittaa, että komento `tiedmenu.add_command(label="Yksi", command=yksi)` lisää Menu-komponenttiin `tiedmenu` valinnan ”Yksi”, johon on kytketty tapahtuma `yksi`. Koodissa on vielä kolmas komento, `add_separator` jolla lisäämme valikkoon erottimia. Tämä komento ei tarvitse parametreja, koska sen ainoa toiminto on lisätä alasvetovalikkoon valintojen väliin tyhjiä välejä helpottamaan valintojen ryhmittelyä. Esimerkiksi kuvan 10 alasvetovalikossa on tällainen välikkö valintojen Kaksi ja Lopeta välissä.

Jos haluamme lisätä toisen alasvetovalikon, tapahtuu se samalla tavoin kuin ensimmäisen luominen. Ensin teemme uuden Menu-komponentin, joka kytketään pohjatason päällä olevaan Menu-komponenttiin. Koodissa myöhemmin määritelty alasvetovalikko sijoitetaan aina viimeisimmän oikealle puolelle.

Lisäksi kuriositeettinä kannattaa huomata, että tällä kertaa tapahtumakutsussa on määritelty käytönaikaisesti tekstikentän päivittäminen uudella tekstillä. Tämän käskyn avulla voit toteuttaa esimerkiksi tekstikenttään ohjetekstin tai yksinkertaisesti esittää käyttäjälle, mitä hän on viimeisimpänä valinnut.

Valintaruudut

Alasvetovalikoiden avulla voimme piilottaa käyttöliittymästä sellaisia valintoja, joita tarvitsemme ainoastaan kun olemme aikeissa tehdä jotain. Entäpä, jos tarvitsisimme mahdollisuuden valita asioita esimerkiksi listalta tai antaa jonkin lisävalinnan, kuten “Muista minut” tai “Tallenna salasana”?

Tämä voidaan on toteuttaa valintaruuduilla. Tkinter-kirjastossa on erillinen komponentti nimeltään Checkbutton, jonka avulla pystymme tekemään valintaruudun, joka saa joko arvon 1 (= valittu) tai 0 (=ei valittu). Seuraavassa esimerkissä tutustumme tämän komponentin toimintaan.

Esimerkki 4.2: Valintaruudut

Esimerkkikoodi

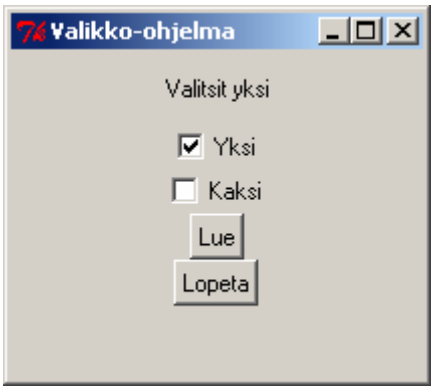
```
# -*- coding: cp1252 -*-
from Tkinter import *
def lue():
    if arvo1.get() == 1 and arvo2.get() == 0:
        ruutu['text'] = "Valitsit yksi"
    elif arvo1.get() == 0 and arvo2.get() == 1:
        ruutu['text'] = "Valitsit kaksi"
    elif arvo1.get() == 1 and arvo2.get() == 1:
        ruutu['text'] = "Valitsit yksi ja kaksi"
    else:
        ruutu['text'] = "Valitse valikoista jotain"

def lopeta():
    root.destroy()
    sys.exit(0)

root = Tk()
root.title("Valikko-ohjelma")
root.geometry("150x150")

frame= Frame(root, width = 240, heigh = 100)
frame.pack(side= TOP)
ruutu = Label(frame, padx = 10, pady = 10, text="Valitse valikoista jotain")
ruutu.pack()

arvo1 = IntVar()
valinta1 = Checkbutton(frame, text="Yksi", variable=arvo1)
valinta1.pack()

arvo2 = IntVar()
valinta2 = Checkbutton(frame, text="Kaksi", variable=arvo2)
valinta2.pack()

luenappi = Button(frame, text = "Lue", command = lue)
luenappi.pack()

lopnappi = Button(frame, text = "Lopeta", command = lopeta)
lopnappi.pack(side = BOTTOM)
mainloop()
```

Esimerkkikoodin tuottama tulos

Kun ajamme ylläolevan lähdekoodin, saamme seuraavanlaisen käyttöliittymäikkunan.

Kuva 14: Esimerkin 4.2 käyttöliittymäikkuna.

Voimme valita jomman kumman tai molemmat valintaruudut. Painaessamme painiketta “Lue”, kertoo käyttöliittymä meille mitä valintaruutuja olemme valinneet.



Kuva 15: Esimerkin 4.2 käyttöliittymä toiminnassa.

Kuinka koodi toimii

Jälleen kerran keskitymme ainoastaan lähdekoodin uusiin asioihin. Ensinnäkin, jos tarkastelemme varsinaista määrittelykoodia, huomaamme käskyn `arvo1 = IntVar()`. Tällä käskyllä luomme muuttujaan `arvo1` Tkinter-moduulin käyttöä varten integer-muuttujan. Tämä toimenpide tehdään sen vuoksi, koska Tkinter-kirjaston komponentit tarvitsevat käyttämänsä muuttujat Tkinterin omissa muuttujatyypeissä. Seuraavalla rivillä luomme käskyllä `valinta1 = Checkbutton (frame, text="Yksi", variable=arvo1)` Checkbutton-komponentin, jolle annamme parametreina Frame-komponentin, johon Checkbutton sijoitetaan ja tekstin, joka tulee valintalaatikon viereen sekä muuttujanarvon, jota Checkbox-komponentti muuttaa tilansa mukaisesti.

Tapahtumakutsussa noudamme käskyllä `arvo1.get()` Tkinter-muuttujan `arvo1` integerarvon. Checkbox-komponentin tila luetaan aina tällä tavoin; muista että et voi suoralla sijoituksella

muuttaa Tkinterin muuttujien arvoja, koska Tkinterin omat muuttujat ovat käytännössä tietorakenteita eikä tavallisia Python-muuttujia. Jos valintaruudussa on rasti, palauttaa funktio `get` arvon 1. Jos ruutu on tyhjä, palautetaan arvo 0.

Yhteenveto

Tässä luvussa tutustuimme hieman erilaisiin Tkinter-kirjaston valintarakenteisiin. Tkinteristä löytyy muita valintarakenteita, kuten Droplist tai RadioButton, mutta ne eivät tällä erää kuulu oppaan aiheisiin. Kuten aikaisemminkin, voit lukea lisätietoa tai tutustua edellä mainittuihin komponentteihin Frederik Lundhin kirjoittamasta verkkokäsikirjasta [2].

Luku 5: Valmiit dialogit

Käyttöjärjestelmän tuki

Aina ei kaikkea kuitenkaan tarvitse rakentaa itse. Esimerkiksi käyttöjärjestelmät tarjoavat usein niin sanottuja vakio-dialogeja kuten vahvistukset, avattavan tiedoston valitsemiset sekä tiedoston tallennuspaikan ja -nimen valinnat. Luonnollisesti myös Tkinter osaa hyödyntää näitä dialogeja kahden apukirjastonsa avulla. Ainakin seuraavat dialogi-ikkunat on saatavilla Tkinter-kirjaston kautta:

- tkMessageBox.showerror: Virheikkuna, ainoa painike OK
- tkMessageBox.showwarning: Varoitusikkuna, ainoa painike OK
- tkMessageBox.showinfo: Tiedoteikkuna, ainoa painike OK
- tkMessageBox.askyesno: Hyväksy/hylkää-ikkuna, painikkeet Kyllä ja Peruuta
- tkMessageBox.askokcancel: Toinen hyväksy/hylkää-ikkuna, painikkeet OK ja Peruuta
- tkFileDialog.askopenfilename: Avaa tiedostonvalintaikkunan
- tkFileDialog.asksaveasfilename: Avaa tiedoston tallennusikkunan

Käytettäessä tkMessageBox-moduulin dialogeja niiden toiminta on aina seuraavanlainen: käsky

```
if tkMessageBox.valinta("Otsikko", "Kysymys?"):
    #tapahtuma jos painetaan OK/Hyväksy/Kyllä
else:
    #tapahtuma jos painetaan Peruuta/Hylkää/Ei (jos vaihtoehtona)
```

Koska `tk.MessageBox`-funktiot palauttavat arvon True (= 1) kun dialogi hyväksytään, laukaisevat ne myös if-rakenteen. Jos taas dialogi hylätään, palautetaan arvo False (= 0). tkFileDialog-funktiot palauttavat joko arvon False tai tiedostokahvan ([1], luku 7) valittuun tiedostoon. Tutustu seuraavaan esimerkkiin, jossa näet kuinka dialogi-ikkunoita varsinaisesti käytetään.

Dialogien käyttäminen

Voidaksemme käyttää dialogeja, tarvitsemme siis käyttöömme kaksi apukirjastoa nimeltään `tkFileDialog`, jos käytämme tiedostonvalintaikkunoita, sekä `tkMessageBox`, jos käytämme muita dialogeja. Seuraavassa esimerkissä tutustumme dialogi-ikkunoiden käyttöön toteuttamalla hieman tavallista pidemmälle toteutetun tekstinkäsittelyohjelman. Lisäksi tutustumme esimerkissä vielä yhteen Tkinterin komponenttiin nimeltä `Text`.

Esimerkki 5.1: Dialogi-ikkunat ja tekstinkäsittelyohjelma

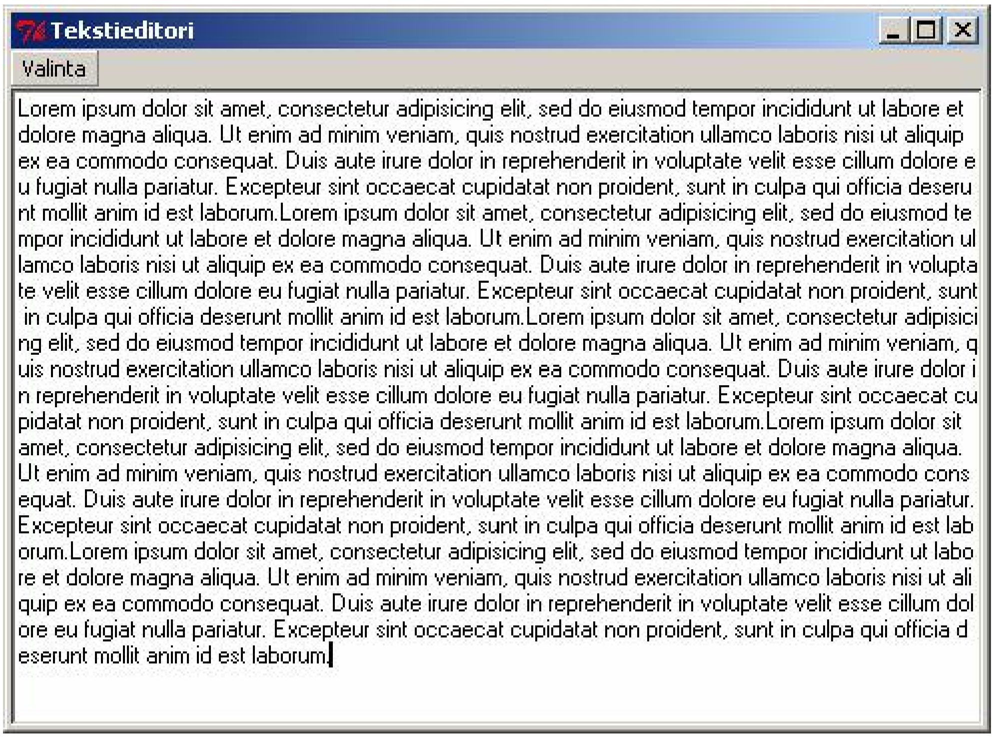
Esimerkkikoodi

```
# -*- coding: cp1252 -*-
from Tkinter import *
import tkMessageBox, tkFileDialog


def lopeta():
    if tkMessageBox.askyesno("Lopetus", "Lopetetaanko?"):
        root.destroy()
        sys.exit(0)

def uusi():
    tkMessageBox.showinfo("Uusi tiedosto", "Luotiin uusi tiedosto")
    teksti.delete(1.0, END)

def avaa():
    tiedostonimi = tkFileDialog.askopenfilename()
    tiedosto = open(tiedostonimi, 'r')
    sisus = tiedosto.read()
    teksti.delete(1.0, END)
    teksti.insert(INSERT, sisus)
    tiedosto.close()

def tallenna():
    tiedostonimi = tkFileDialog.asksaveasfilename()
    tiedosto = open(tiedostonimi, 'w')
    sisus = teksti.get(1.0, END)
    tiedosto.write(sisus)
    tiedosto.close()


root = Tk()
menu = Menu(root)
root.title("Tekstieditori")
root.config(menu=menu)

tiedostomenu = Menu(menu)
menu.add_cascade(label="Valinta", menu=tiedostomenu)
tiedostomenu.add_command(label="Uusi", command=uusi)
tiedostomenu.add_command(label="Avaa...", command=avaa)
```

```
tiedostomenu.add_command(label="Tallenna...", command=tallenna)
tiedostomenu.add_separator()
tiedostomenu.add_command(label="Lopeta", command=lopeta)

frame= Frame(root, width = 240, heigh = 100)
frame.pack(side= TOP)

teksti = Text(frame)
teksti.pack(expand = YES)

mainloop()
```

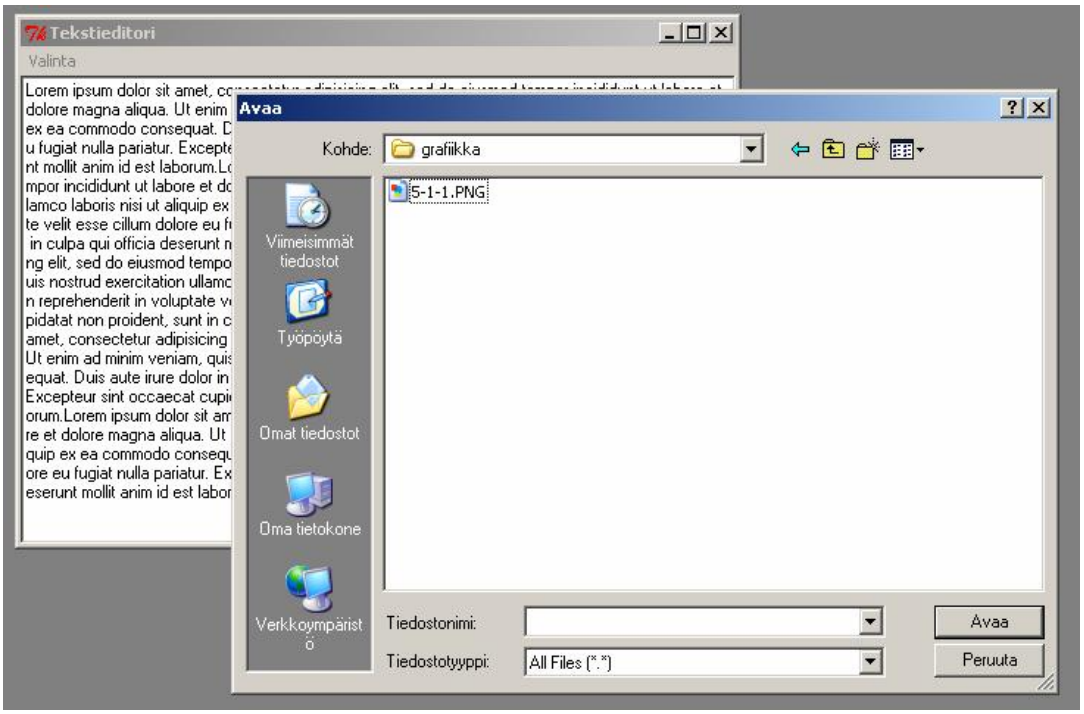
Esimerkkikoodin tuottama tulos

Kun ajamme esiemerkin lähdekoodin saamme aikaan seuraavanlaisen käyttöliittymäikkunan (teksti lisätty jälkikäteen).



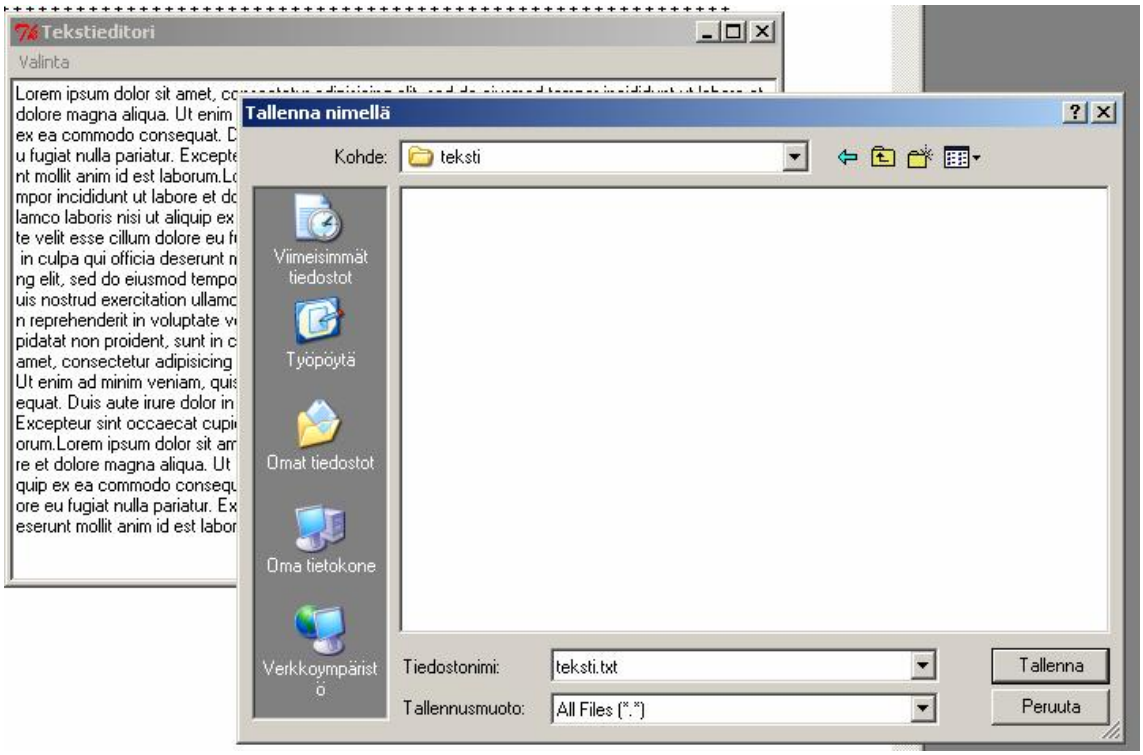
Kuva 16: Esimerkin 5.1 luoma käyttöliittymäikkuna.

Kun valitsemme alasvetovalikosta valinnan “Avaa”, saamme seuraavanlaisen dialogi-ikkunan.



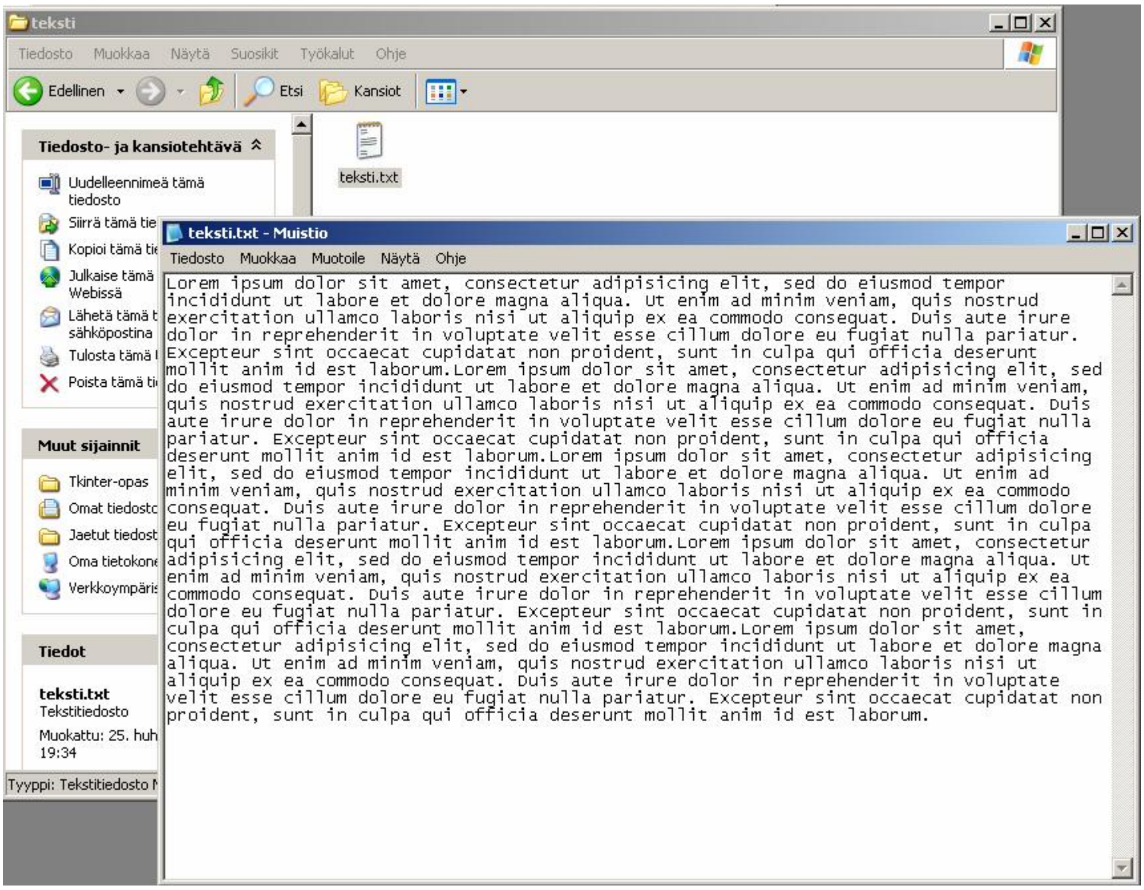
Kuva 17: Esimerkin 5.1 Avaa-dialogi.

Kun valitsemme alasvetovalikosta valinnan “Tallenna”, saamme seuraavanlaisendialogi-ikkunan.



Kuva 18: Esimerkin 5.1 Tallenna-dialogi

Jos tarkastelemme tallentamaamme tiedostoa vaikkapa käyttöliittymän omalla tekstieditorilla, huomaamme että ohjelmamme toimii.

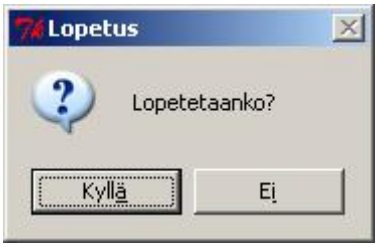


Kuva 19: Esimerkin 5.1 tallentama työ Muistio-editorissa.

Lisäksi voimme valita alasvetovalikosta vaihtoehdon ”Uusi”, joka tyhjentää tekstikentän ja tuottaa seuraavanlaisen dialogi-ikkunan:

Kuva 20: Esimerkin 5.1 Uusi tiedosto-dialogi.

Voimme myös valita ”Lopeta”, joka ensin esittää seuraavanlaisen valintadialogin ja tarvittaessa lopettaa ohjelman.



Kuva 21: Esimerkin 5.1 lopetusdialogi.

Kuinka koodi toimii

Kun aloitamme tarkastelemalla lähdekoodin päätasoa, huomaamme että siellä on määritelty uusi komponentti `Text`. Text-komponentti tarkoittaa tekstikenttää, johon voimme kirjoittaa halutessamme useita rivejä tekstiä ilman että joudumme määrittelemään jokaista tekstiriviä erikseen. Lisäksi Text-komponentti huolehtii itse rivityksestä sekä jonkinlaisesta varatun alueen yli jatkuvuudesta. Tärkeitä tässä esimerkissä on kuitenkin se, että ymmärrät kuinka tekstikenttä voidaan luoda. Lisäksi tekstikentän pakkauksen yhteydessä sille on annettu `expand`-parametrilla ominaisuus laajentua siten, että se kattaa koko sen käyttämän käyttöliittymäikkunan tilan huolimatta siitä, että käyttäjä hiirellä muuttaisi ikkunan kokoa.

Varsinaiset dialogien käyttöön liittyvät koodit on sijoitettu tapahtumakutsuihin. Ensinnäkin lopetuskäskyn varmistava dialogi toimii yllä esitellyn if-rakenteen avulla. Ainoastaan mikäli käyttäjä hyväksyy lopetuksen, lopetamme ohjelman. Muuten meillä ei ole tarvetta tehdä mitään, joten erillistä else-osiota ei tässä tapauksessa tarvita. Samoin uuden tiedoston luomisen yhteydessä ainoastaan annamme dialogin yllä olevan esimerkkikoodin mukaisesti ja tyhjennämme Text-komponentin poistamalla merkit 0:sta END:iin, eli ensimmäisestä merkistä viimeiseen. Tiedostoja käsiteltäessä dialogi palauttaa normaalin tiedostokahvan, jota käytetään kuten tavallisestikkin. Tiedostokahvasta ja tiedostoon itseensä kirjoittamisesta voit lukea tarkemmin perusohjelmointioppaasta [1].

Yhteenveto

Tässä luvussa tutkimme käyttöjärjestelmän tarjoamia palveluja käyttöliittymälle. Kuten huomaat, ei kaikkea tarvitse aina rakentaa itse, vaan usein riittää, että tiedämme mitä haluamme. Tässä ei tietenkään esitelty kaikkia Tkinter-moduulin valmiita dialogeja vaan ainoastaan niistä tavallisimmat. Lisää dialogeista voit lukea Lundhin kirjoittamasta verkkokäsikirjasta [2].

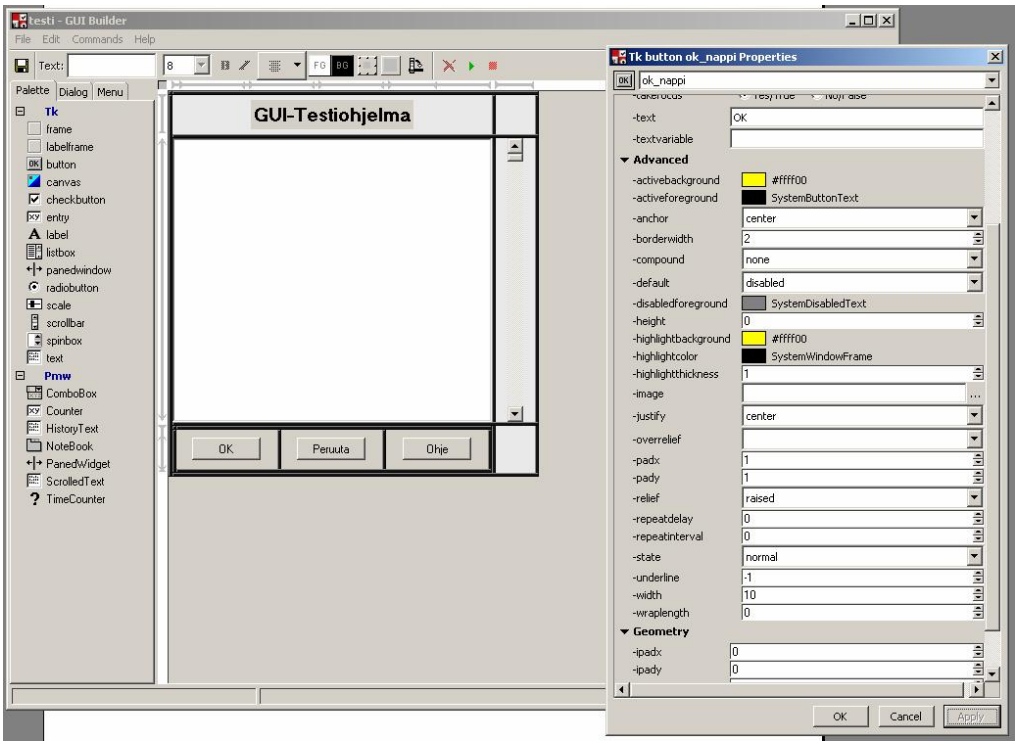
Luku 6: Visuaalisista editoreista sekä suunnittelusta

Johdanto

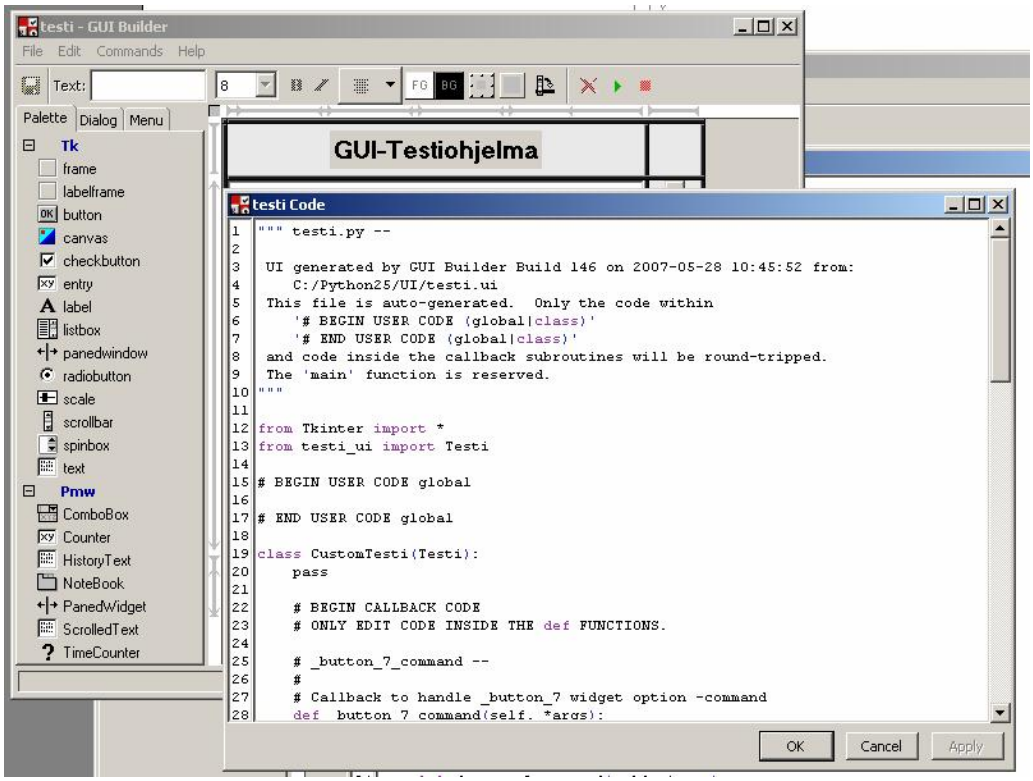
Tässä oppaassa olemme tutustuneet tapaan tuottaa käyttöliittymiä tekemällä ensin suunnitelma käyttöliittymän toteutuksesta ja tämän jälkeen kirjoittamalla valmista koodia, joka tuottaa halutunlaisen käyttöliittymän. Kun puhuimme aiemmin siitä, kuinka ohjelmien käyttöliittymän tuottaminen voi joissain tapauksissa vaatia ammattilaistason työkaluja, tarkoitimme niillä mm. visuaalisia editoreita. Näiden ohjelmien avulla voimme luoda käyttöliittymäikkunoita ”maalaamalla” pohjatason päälle halutunlaisen käyttöliittymän. Tämä siis tarkoittaa sitä, että sen sijaan että kirjoittaisimme koodia, voimme valita esimerkiksi tekstikentät, painikkeet sekä valintaruudut listalta ja tiputtaa ne hiirellä halutuille paikoille. Lopuksi, kun olemme tyytyväisiä aikaansaamaamme käyttöliittymään, voimme pyytää ohjelmaa toteuttamaan tehdyn käyttöliittymän toimivana koodina.

Visuaalinen editori Tkinterille

Myös Tkinter-käyttöliittymä voidaan suunnitella visuaalisen editorin avulla. Esimerkiksi Activestate on toteuttanut GUI Builder-nimisen käyttöliittymän suunnitteluun tarkoitetun visuaalisen editorin, joka voit ladata ilmaiseksi verkosta [4]. Huomioi, että ohjelma on alun perin suunniteltu Tcl-kielen Tk-käyttöliittymäohjelmointiin, mutta sillä voidaan tuottaa myös Python/Tkinter-koodia valitsemalla Python/Tkinter-käännöskieleksi alkuvalikossa. Ohjelman avulla voimme luoda hiiren avulla käyttöliittymäikkunoita sekä muutella jo luotujen komponenttien parametreja. Editori on niin yksityiskohtainen, että pystymme luomaan sillä käyttöliittymiä WYSIWYG (what you see is what you get)-periaatteella. Tämä tarkoittaa sitä, että generoidulla koodilla tuotettu käyttöliittymä on aina täsmälleen samanlainen kuin se, miltä käyttöliittymä visuaalisessa editorissa näytti. Tämä ei kuitenkaan tarkoita sitä, että editorilla olisi mahdollista toteuttaa valmiita ohjelmia. Generoitu koodi onkin pelkkä käyttöliittymä; tapahtumat, toiminnallisuus sekä tietorakenteet joudutaan toteuttamaan jälkikäteen käsin. Koska ohjelma on hyvin laaja sekä siihen on olemassa projektin sivujen [4] kautta kattavat käyttöohjeet, emme tässä oppaassa puutu sen tarkemmin ohjelman käyttämiseen. Ohessa muutamia kuvia visuaalisesta editorista käytössä.



Kuva 22: GUI Builder käytössä, vasemmalla varsinainen editori, oikealla valitun komponentin asetuksia ja parametreja.



Kuva 23: Generoitu käyttöliittymäkoodi; tästä koodaus jatkuisi toiminnallisuuksien luomisella IDLE-editorin puolella.

Huomioita suunnittelusta

Käyttöliittymäeditoreista yleisesti

Nykyisin ohjelmien graafisen käyttöliittymän toteuttamisen voidaan sanoa tapahtuvan helpoiten juuri visuaalisten editorien avulla. Nämä ohjelmat nopeuttavat monessa tapauksessa varsinaisen käyttöliittymän suunnittelua huomattavasti verrattuna käsin tehtyyn käyttöliittymään, mutta ne eivät vielä takaa onnistuneen käyttöliittymän toteuttamista. Monesti visuaalisella editorilla luotu käyttöliittymä tehdään helposti liian sekavaksi, koska käyttöliittymän tekeminen itsessään on hyvin vaivatonta. Lisäksi editorilla luotu koodi on usein parhaimmillaankin sekavaa verrattuna itse kirjoitettuun koodiin, joten virheiden määrä kirjoitetussa koodissa kasvaa. Eikä myöskään tule unohtaa sitä, että generoitu koodi voi editorista riippuen kasvaa helposti räjähdysmäisesti; esimerkissä käytetty koodi on jo nyt yli 100 riviä koodia, vaikka olemme luoneet vasta yhden ikkunan, jossa siinäkään ei ole minkäänlaista toiminnallisuutta.

Tyylioppaista

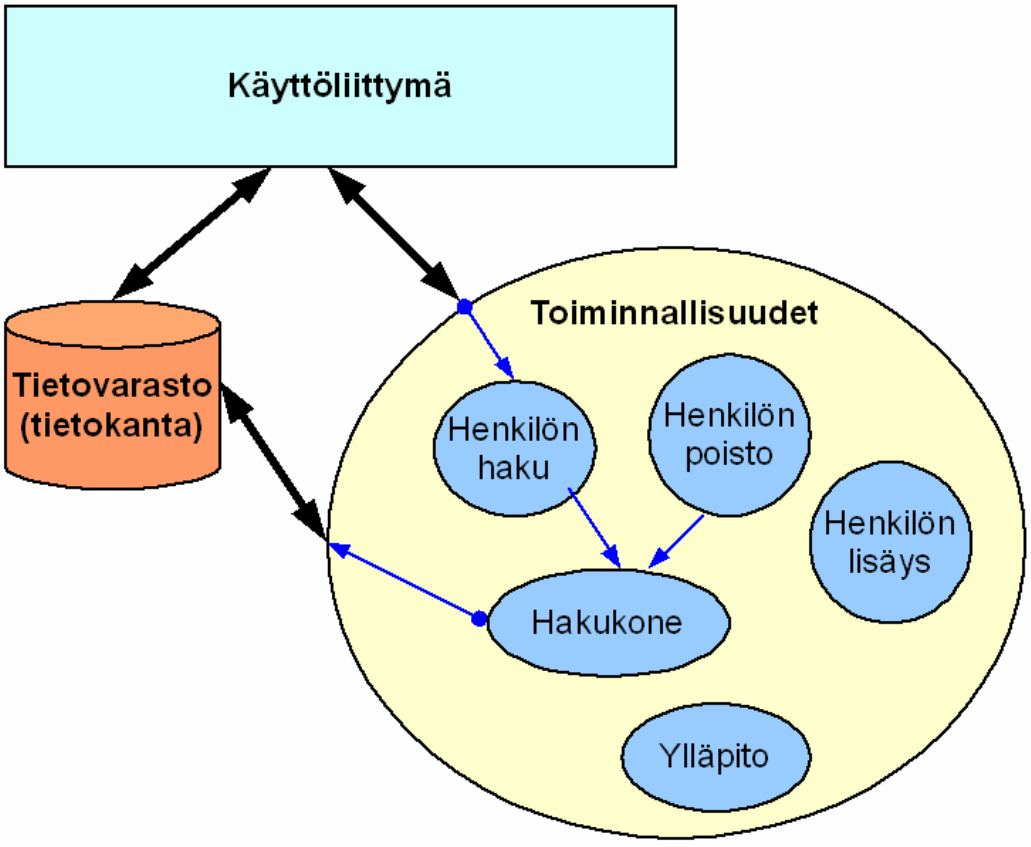
Käyttöliittymän suunnittelussa on tavallisesti käytössä tyylioppaat, joiden avulla käyttöliittymän suunnittelulle asetetaan suuntaviivat, joiden mukaan ohjelmoijien on tarkoitus toimia. Esimerkiksi Windows XP-käyttöjärjestelmää varten on olemassa yleinen tyyliopas [5], jossa määritellään mm. miltä Windows XP-tyylisen ohjelman tulee näyttää, kuinka sen tulee toimia ja mitä esimerkiksi ”OK”-painikkeista tulee yleisesti ottaen tapahtua. Näiden oppaiden avulla pyritään yhdenmukaistamaan ohjelmien ulkonäköä sekä tekemään ohjelmien käyttäminen asiakkaille helpoksi. Luultavasti jokaisella vakavasti otettavalla ohjelmistoalan yrityksellä onkin oma tyyliopas tai se käyttää ohjelmissaan jotain valmista opasta. Mikäli siis olet aloittamassa suurikokoista (10000+ riviä ohjelmakoodia) projektia, on myös sinun hyvä miettiä etukäteen yleiset tyyliseikat valmiiksi, jotta ohjelmastasi tulee ammattimaisen ja siistityn näköinen ja että sinä tai joku muu pystyy myöhemmässä vaiheessa ylläpitämään tekemääsi ohjelmakoodia.

Tyylioppaat eivät myöskään rajoitu pelkästään käyttöliittymän suunnitteluun. Kuten varmaan huomasit, on visuaalisen editorin tuottama koodi hyvin kommentoitua sekä täynnä merkintöjä ja mainintoja siitä, mitä mistäkin tapahtuu. Jo aiemmissa oppaissa [1] olemme puhuneet siitä, kuinka oman lähdekoodin kommentointi kannattaa, vaikka sen tekeminen voikin tuntua puuduttavalta. Kirjoittamistasi kommenteista ja huomautuksista onkin hyötyä viimeistään silloin, kun kuukauden päästä tutkit omaa koodiasi aikomuksenasi muokata tai korjata sitä. Jos alat suunnittelemaan suurikokoista projektia, on sinun hyvä käyttöliittymän lisäksi suunnitella tyyliopas lähdekoodillesi. Tästä tyylioppaasta tulisi mm. selvitä funktioiden ja muuttujien nimeämiskäytännöt, parametrien syöttöjärjestyksen mahdollinen standardointi, vakioidut muuttujannimet, kommentointien käyttö ja sijainti, mahdolliset ohjerivit sekä muutenkin ohjelmakoodin rakenne. Suosittelemme, että tutustut esimerkiksi Python-ohjelmointikielen luojan, Guido van Rossumin, kirjoittamaan Python-tyylioppaaseen [6] ja mietit, voisitko itse parantaa koodisi rakennetta ja tehdä siitä tyylioppaan

avulla johdonmukaisempaa ja selkeämpää. Koska viimeistään toteuttaessasi suurempaa ohjelmaa tai ylläpitäessäsi olemassa olevaa ohjelmakoodia tulet hyötymään johdonmukaisesta ja selkeästä koodaustyylistä.

Modulaarisuudesta

Vielä kolmas ohjelmien suunnittelussa huomioon otettava asia on modulaarisuus. Tällä tarkoitetaan sitä, että teollisissa ohjelmaprojekteissa (100000+ riviä koodia) ohjelmat tavallisimmin suunnitellaan siten, että ohjelman toiminnalliset osat on hajautettu eri kokonaisuuksiin eli moduuleihin. Yleisimmin toisistaan erotettavia osia ovat ainakin tietovarasto – mihin tiedot on fyysisesti tallennettu -, toiminnallisuudet – operaatiot joilla muokataan tietovaraston tietoja – sekä käyttöliittymä. Etuna tällaisessa lähestymistavassa on se, että ohjelman yksittäinen osa voidaan tarvittaessa korvata toisella ilman että koko ohjelma joudutaan suunnittelemaan uudelleen.



Kuva 24: Modulaarinen ohjelman rakenne

Tämä tietysti tarkoittaa silloin myös sitä, että ohjelman eri osa-alueiden välille tulee suunnitella rajapinnat, joiden avulla ohjelman osat vaihtavat tietoa. Käytännössä tämä tarkoittaa sitä, että ohjelmat suunnitellaan siten, että yksittäinen ohjelman osa on vaikkapa erillinen moduuli, jonka funktioita kutsutaan jollain etukäteen päätetyillä parametreilla ja se palauttaa uudet arvot aina tietyssä muodossa. Tällöin kutsuvan ohjelman ei tarvitse tietää miten moduuli uudet arvot sai, vaan sille riittää tieto siitä että lähettämällä tietynlaista dataa moduulin funktiolle X se saa aina tietynlaisen vastauksen. Yksin koodatessasi tämä ei usein muodostu kynnyskysymykseksi, mutta

laajemmissa projekteissa voi olla, että ohjelman eri osa-alueiden - jopa eri laskentafunktioiden tai toiminnallisuuksien – toteutuksesta vastaa täysin erilliset ihmiset joilla ei välttämättä edes ole kommunikaatioyhteyttä keskenään. Jos tarkastelemme kuvaa 24, voimme hieman tarkemmin perehtyä tähän ajatukseen. Jos esimerkiksi haluaisimme toteuttaa kuvan kaltaisen henkilötietoja hallinnoivan ohjelman, olisi meidän kannattavaa toteuttaa ohjelma modulaarisesti. Ajatelkaamme esimerkiksi ohjelmaan suunniteltua hakukonetta: jos keksimme jossain vaiheessa keinon hakea tietoja nopeammin tai tarkemmin tietovarastosta, ei meidän tarvitse kuin korvata itse *Hakukone*-toiminnallisuus. Ohjelmamme toimii edelleen, mikäli vain huolehdimme, että toiminnallisuus *Henkilön haku* saa palautusarvoina sellaisia parametreja, joita se pystyy hyödyntämään. Samoin olisi esimerkiksi ylläpitoon toimintojen lisääminen. Voisimme luoda uuden toiminnallisuuden erillään jo olemassa olevista toiminnoista, ja lisätä *Ylläpito*-toiminnallisuuteen pelkän käynnistyskäskyn, jolla toiminnallisuus tapahtuisi. Tämän vuoksi onkin tärkeää suunnitella ohjelma osakokonaisuuksina, jotka kommunikoivat keskenään etukäteen sovittujen rajapintojen yli.

Loppusanat

Huomioita

Kuten alussa totesimme, ei tämän oppaan tarkoitus olekaan tarjota täydellistä pakettia jonka avulla tutustuisimme kaikkiin Tkinter-käyttöliittymätyökalun aspekteihin, vaan mahdollistaa se, että voit jatkossa ymmärtää Tkinter-koodin rakennetta paremmin. Tietenkin tulee muistaa, että tulevaisuudessa on täysin mahdollista jatkaa tai laajentaa tämän oppaan sisältöä myöhemmissä versioissa, mutta tällä erää opas on nimenomainen läpileikkaus graafisen käyttöliittymän toteuttamisesta. Siihen asti kuitenkin toivomme, että tästä ”alustuksesta” on ollut sinulle hyötyä ja että saat sen avulla toteutettua tarpeidesi mukaisia työkaluja ja käyttöliittymiä, sekä pääset alkuun, mikäli aiot toteuttaa jonkin suuremman ohjelmaprojektin Pythonilla.

Lisäluettavaa

Kuten tekstissä on useampaan kertaan mainittu, on Fredrik Lundhin Tkinter-opas [2] erinomainen lähdeteos komponentteihin tutustumiseen. Lundhin opas on englanninkielinen, eikä se sisällä varsinaista opetusmateriaalia tutoriaalin tai funktionaalisten esimerkkikoodien muodoissa. Kuitenkin verkosta löytyy myös muita Tkinter-oppaita, jotka on tarkoitettu myös alkeiden opiskeluun. Näitä on mm. Stephen Fergin ”Thinking in Tkinter” [7] tai Shipmanin ”Tkinter: GUI programming with Python” [8]. Kolmas erinomainen lähde, missä voit tutustua Tkinter-ohjelmien tekemiseen tarkemmin on Tkinter Wiki [9], josta löytyy sivun omien esimerkkien ja ohjeiden lisäksi paljon jatkolinkkejä muihin ohjeisiin. Voit tietenkin myös jatkaa Tkinter-ohjelmointiin tutustumista hyödyntämällä GUI Builder-editoria sekä sen dokumentaatioita [4]. Lisäksi, mikäli aiot vakavissasi alkaa toteuttamaan Pythonilla oikeita Windows-ohjelmia, kannattaa sinun myös tutustua tämän oppaan liitteessä A esiteltyyn py2exe-nimiseen Windows-paketoijaan [10], jolla voit tehdä tulkista riippumattomia ohjelmia. Lisäksi, mikäli olet kiinnostunut grafiikkaohjelmoinnista tai kuvankäsittelystä Python-ohjelmointikielen avulla, voit tutustua aiemmin ilmestyneeseen Python-grafiikkaohjelmointioppaaseen [11].

Lähdeluettelo

[1] Kasurinen, Jussi (2006). Python-ohjelmointiopas, versio 1. Lappeenrannan teknillinen yliopisto, Tietotekniikan käsikirjat 7. ISBN 952-214-286-7

[2] Lundh, Fredrik (1999). An Introduction to Tkinter. Saatavilla osoitteesta http://www.pythonware.com/library/tkinter/introduction/index.htm. Viitattu 10.5.2007

[3] Python Software Foundation (2007) www.python.org

[4] SpecTCL GUI Builder (2007) http://spectcl.sourceforge.net/

[5] Microsoft Corporation, Windows XP User Experience Guidelines versio 1.0a (2007) Saatavilla verkosta sivujen www.microsoft.com kautta.

[6] van Rossum, Guido (2007) Style Guide for Python Code. Saatavilla verkosta osoitteesta http://www.python.org/dev/peps/pep-0008/

[7] Freg, Stephen (2006), Thinking in Tkinter. Saatavilla osoitteesta http://www.ferg.org/thinking_in_tkinter/index.html, viitattu 15.5.2007

[8] Shipman, John (2005), Tkinter: GUI programming with Python. Saatavilla osoitteesta http://infohost.nmt.edu/tcc/help/lang/python/tkinter.html, viitatty 15.5.2007

[9] Tkinter Wiki (2007), http://tkinter.unpythonic.net/wiki/

[10] py2exe-projektin kotisivut (2007). Saavilla osoitteesta http://www.py2exe.org, viitattu 21.5.2007

[11] Kasurinen, Jussi (2007). Python –grafiikkaohjelmointi Imaging Librarylla. Lappeenrannan teknillinen yliopisto, Tietotekniikan käsikirjat 8. ISBN 978-952-214-386-0

Liite A: py2exe-paketoija

Itsenäisen ohjelman tekeminen

Tässä oppaassa olemme tutustuneet siihen, kuinka Python-ohjelmointikielellä voidaan toteuttaa graafisen käyttöliittymän avulla ”oikeita” tietokoneohjelmia. Nyt kun osaamme valmistaa erilaisia työkaluja ja pieniä ohjelmia – kuten luvun 5 tekstinkäsittelyohjelma – olisi meidän hyvä tarkastella ohjelman suorittamista hieman tarkemmin.

Kuten varmaan olet huomannut, joudut asentamaan koneellesi Python-tulkin, jotta voit suorittaa kirjoittamasi lähdekoodin työasemallasi. Tämän joudut tekemään juuri sen vuoksi, että tarvitset koneeseen Python-tulkin, joka tulkkaa kirjoittamasi Python-koodin tietokoneen ymmärtämään muotoon. Tämä rajoite ei ohjelmoijan kohdalla ole mitenkään merkityksellinen, mutta entäpä, jos luot ohjelman jota haluaisit levittää muiden ihmisten saataville? Koska emme voi odottaa ihmisten haluavan asentaa koko Python-ohjelmointiympäristöä vain sinun ohjelmasi takia, joudumme tekemään koodista itsenäisen kokonaisuuden.

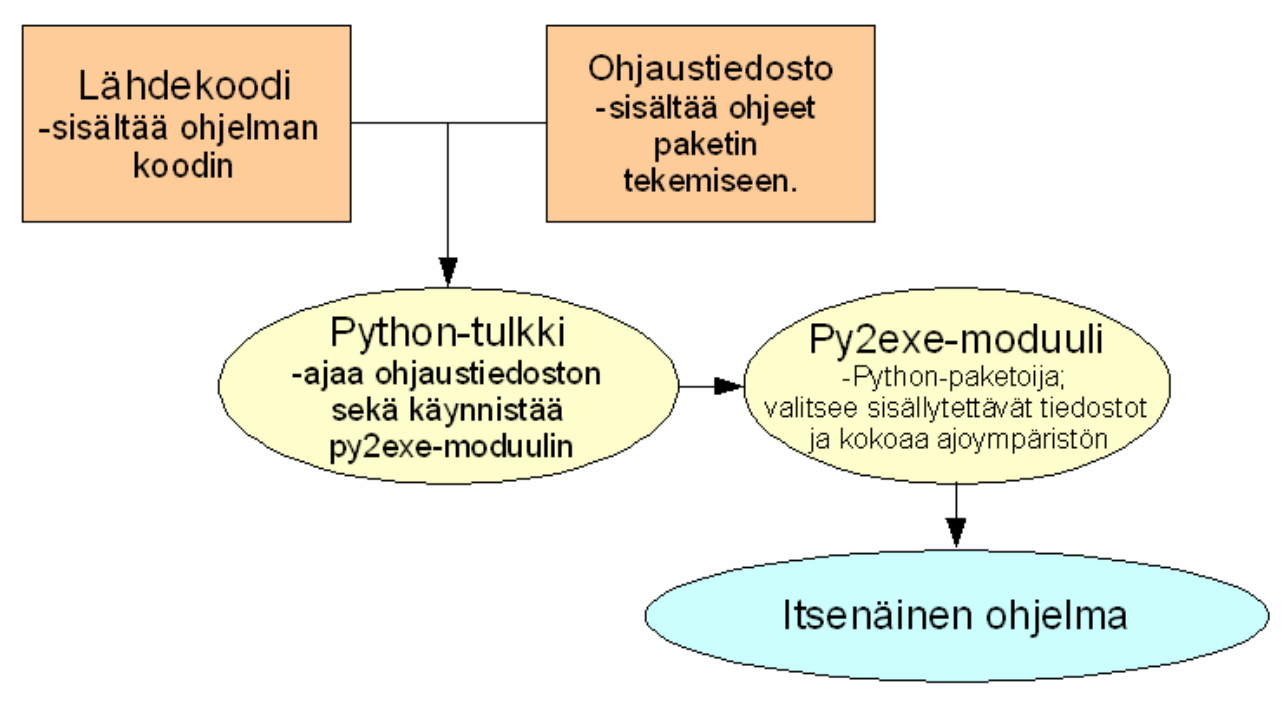
Tällä tarkoitetaan ohjelmakoodin tilaa, jossa tulkin ymmärtämästä lähdekoodista on toteutettu pysyvä itsenäisesti suoritettavissa oleva versio. Tätä versiota et useimmiten enää pysty itse muokkaamaan, mutta toisaalta et myöskään tarvitse erillistä ajoympäristöä koneissa, joissa haluat koodisi olevan ajettavissa. Python-ohjelmointikielelle on olemassa muutamia menetelmiä, joilla Windows-puolella pystymme luomaan itsenäisiä ohjelmia. Näistä ehkäpä yksinkertaisin käytettävä on nimeltään py2exe. Tämä työkalu toimii siten, että se tuotattaa tulkilla kirjoittamastasi lähdekoodista ja sen käyttämistä kirjastomoduuleista esikäännetyt versiot ja paketoi kokonaisuuteen mukaan Python-tulkin toiminnalliset osat. Tällöin kirjoittamastasi lähdekoodista on luotu itsenäinen ohjelma, joka voidaan ajaa ilman käyttöjärjestelmään asennettua erillistä Python-ohjelmointiympäristöä tai -tulkkia.

Valmistelut

py2exe on erikseen asennettava lisämoduuli, jonka asennuspaketti sinun tulee ensin hakea verkosta ja tämän jälkeen asentaa työasemallesi. Kun olet hakenut asennuspaketin, joka löytyy projektin sivuilta osoitteesta http://www.py2exe.org/ [10], joudut vielä asentamaan sen käsin. Asennusvaiheet ovat samat kuin muillakin Python-ohjelmointiympäristön lisämoduuleilla; voit halutessasi hakea tarkemmat asennusohjeet esimerkiksi Python-grafiikkaohjelmointioppaasta [11], jossa asennetaan vastaavilla asennusvaiheilla Python Imaging Library.

Paketoinnin toteuttaminen

py2exe toimii hieman eri tavoin kuin aiemmin käyttämämme Python-ohjelmat. py2exe-moduulia ei käytetä kuten tavallisia kirjastomoduuleja, eli sisällyttämällä kirjasto koodiin, vaan luomalla erillinen ohjaustiedosto (*make-file*). Tällä ohjaustiedostolla ohjaamme py2exe-paketoijan toimintaa antamalla sille tietoina muun muassa haluttu käyttöliittymätyyppi, paketoitavan lähdekoodin nimi sekä lisäparametrit, kuten tuotettavan ohjelman kuvaketiedoston nimi. Kuvaan 25 on kuvattu kuinka py2exe-paketoija toimii. Vaikka py2exe periaatteessa onkin vain yksi Python-tulkin lisämoduuli, on se siinä mielessä erilainen kuin muut koska tulkkia käytetään ainoastaan paketointimoduulin käynnistämiseen ja ajamiseen. Toisin kuin tavallisesti, varsinainen toiminnallisuus tapahtuu pääosin py2exe:n itsensä sisällä.



Kuva 25: py2exe-paketoijan toimintamalli.

Kun ajamme ohjaustiedoston Python-tulkilla samasta kansiosta kuin mihin lähdekooditiedosto on tallennettu, suorittaa py2exe-paketoija kokoamistyönsä ja luo hakemistoon alihakemiston, johon koottu ohjelma tallennetaan. Seuraavilla kahdella esimerkillä demonstroimme paketoijan toiminta aluomalla viidennen luvun tekstieditorista itsenäisesti ajettavan ohjelman. Molemmissa esimerkkitapauksissa ohjelman lähdekoodi on tallennettu tiedostoon nimeltä *lahdekoodi.py*. Paketointiprosessi suoritetaan ajamalla ohjaustiedosto komentoriviltä (kts. lisäohjeet perusopas [1] luku 8) komennolla

```
[ASENNUSPOLKU]\python ohjaus.py py2exe
```

jossa `[ASENNUSPOLKU]` on Python-tulkin asennuskansio (oletus C:\Python25\) sekä `ohjaus.py` ohjaustiedoston nimi.

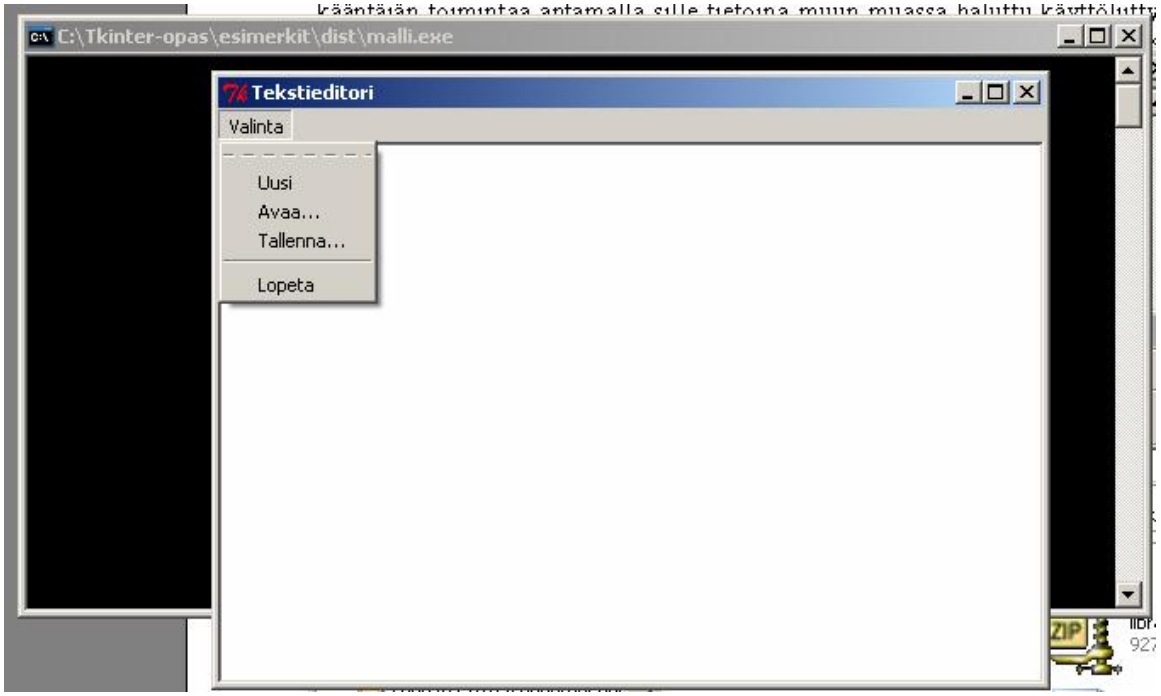
Esimerkki A.1: Komentokehote-pohjainen ohjelma

Paketoijan ohjaustiedoston *ohjaus.py* sisältö

```
from distutils.core import setup
import py2exe

setup(console=['lahdekoodi.py'])
```

Esimerkkikoodin tuottama tulos

Kun suoritamme paketoinnin yllä olevan kaltaisella ohjaustiedostolla ajamalla sen komentoriviltä, saamme aikaan alihakemiston nimeltä *dist*. Tästä kansiosta löytyy tiedosto *lahdekoodi.exe*, jonka ajamalla saamme aikaan seuraavanlaisen esityksen:



Kuva 26: Ensimmäinen py2exe-ohjelma.

Lisäksi ohjelma tulostaa komentokehoitteen ikkunaan useita rivejä tekstiä sekä luo toisen kansion nimeltä *build*.

Kuinka koodi toimii

Emme tällä kertaa keskity niinkään siihen, miten py2exe teknisesti paketoinnin tekee vaan siihen, mitä sen yhteydessä tapahtuu. Tarkastelkaamme ensimmäisenä py2exen rakennustiedostoa.

Kahdella ensimmäisellä rivillä otamme käyttöömme paketointia varten tarvittavat moduulit `distutils.core` ja `py2exe`. Tämän jälkeen kolmannella rivillä määräämme asetuksiin, että haluamme luoda konsolipohjaisen ohjelman lähdetiedoston *lahdekoodi.py* koodista sekä sen käyttämistä lisämoduuleista.

Kun py2exe alkaa työskentelemään lähdekoodin kanssa, se päättelee ensin mitkä kaikki Python-tulkin kirjastot sen täytyy sisällyttää pakettiin. Tämän jälkeen ohjelma tuotattaa tulkilla kirjastot sekä lähdekoodin esikäännettyyn muotoon ja luo kansiota nimeltään *dist* (lopulliset tiedostot) ja *build* (käännöksen aikaiset tiedostot). Paketoija kokoaa ja kääntää tarvittavat kirjastot hakemistossa *build* ja lopuksi siirtää ne virtuaalisen ajoympäristön kanssa hakemistoon *dist*. Kun työ on valmis, sisältää hakemisto *dist* valmiin ohjelman, joka käynnistetään alkuperäisen lähdekooditiedoston mukaan nimetystä exe-tiedostosta. Lisäksi kansiosta *dist* löytyy muita ohjelman ajamisessa tarvittavia tiedostoja tavallisesti useissa eri alikansioita. Mikäli haluat pakata tekemäsi ohjelman esimerkiksi yhdeksi zip-paketiksi, muista sisällyttää kaikki kansiossa *dist* olevat tiedostot ja alihakemistot mukaan pakettiin. **Älä uudelleennimeä tai poista yhtäkään tämän kansion tiedostoa!** Voit kuitenkin muuttaa *dist* -hakemiston nimen mieleiseksesi. Kansio *build* sisältää ainoastaan paketoinnin aikana käytettyjä tiedostoja ja kansion voikin poistaa käännöksen valmistuttua.

Kun tutkimme itse ohjelmaa niin huomaamme, että aina ajaessamme ohjelman *lahdekoodi.exe*, aukeaa myös komentokehotteen ikkuna vaikka emme käytä sitä mihinkään. Tämä johtuu nimenomaan rakennustiedoston määritteestä ”console”, joka tarkoittaa että ohjelmaa varten avataan aina komentokehotteen ikkuna. Koska olemme tehneet koodiimme graafisen käyttöliittymän emmekä tarvitse komentokehotetta, emme myöskään haluaisi sen näkyvän suorituksen yhteydessä. Lisäksi voimme määritellä Windows-ohjelman käyttämän kuvakkeen, jolla voimme visuaalisesti tunnistaa ohjelmamme. Tämä kuvaketiedosto on lähdekoodimme kanssa samassa hakemistossa nimellä `kuvake.ico`.

Kuva 27: Käytettävä kuvake kuvake.ico

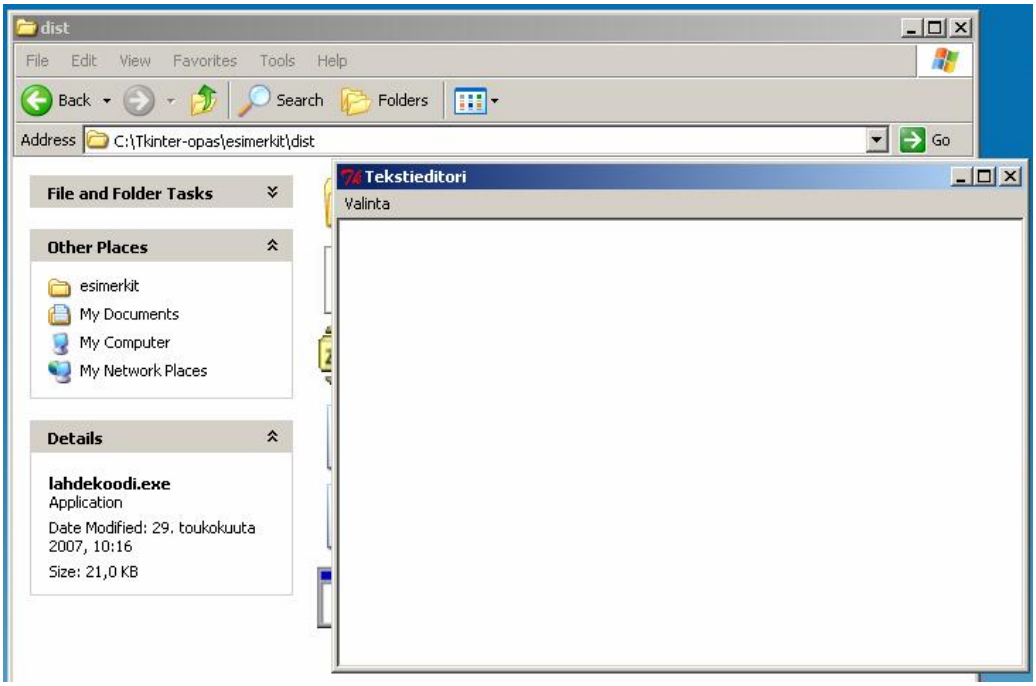
Esimerkki A.2: Windows-ohjelma

Paketoijan ohjaustiedoston *ohjaus.py* sisältö

```
from distutils.core import setup
import py2exe

setup(
    windows = [
        {
            "script": "lahdekoodi.py",
            "icon_resources": [(1, "kuvake.ico")]
        }
    ],
)
```

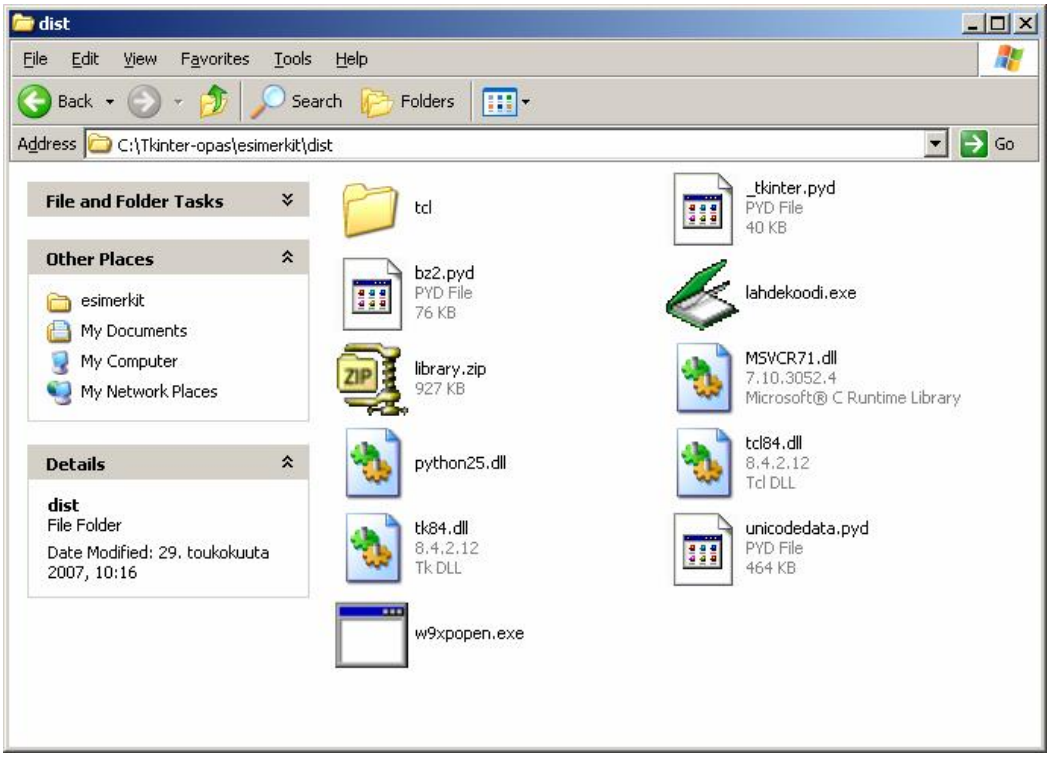
Esimerkkikoodin tuottama tulos

Kun suoritamme paketoinnin yllä olevan kaltaisella ohjaustiedostolla ajamalla sen komentoriviltä käskyllä `python ohjaus.py py2exe`, saamme aikaan alihakemiston nimeltä *dist*. Tästä kansiosta löytyy tiedosto *lahdekoodi.exe*, jonka ajamalla saamme aikaan seuraavanlaisen esityksen:



Kuva 28: Toinen py2exe-ohjelma. Tällä kertaa ohjelma ei avaa komentokehote-ikkunaa.

Lisäksi, jos tarkastelemme kokonaisuutta kansiossa *dist*, huomaamme että py2exe on sisällyttänyt valitsemamme kuvakkeen käynnistävän exe-tiedoston oletuskuvakkeeksi (kuva 29):



Kuva 29: Kerätyt tiedostot. Huomaa kuvake lahdekoodi.exe -tiedostolla.

Kuinka koodi toimii

Tarkastelkaamme ensin eroja ensimmäisen ja toisen ohjaustiedoston välillä. Ensinnäkin, tällä kertaa parametri `console` on vaihtunut parametriksi `windows`. Tällä annamme paketoijalle ohjeen toteuttaa ohjelmasta MS-Windows-ohjelma, jota ajettaessa komentokehoteikkunaa ei näytetä. Lisäksi annoimme parametrilla `icon_resources` haluamamme kuvakkeen, jonka py2exe lisäsi exe-tiedostolle. Tässä kohtaa annoimme arvoina kuvaketiedoston nimen, sekä järjestysnumeron, monesko kuvake haluamamme kuvake tiedostossa oli. Koska ico-tiedostot eivät voi sisältää kuin yhden kuvakkeen, oli valintamme syötteen arvolle luonnollisesti 1.

Rakenteellisesti koodi on edelleen hyvin yksinkertainen. Otamme käyttöön paketoinnissa tarvittavat kirjastot ja tämän jälkeen kutsumme paketoijaa `setup`-funktiolla. Koska `setup`-funktio saa paketoinnin määrittelystä johtuen paljon parametreja ja sisältää sisäkkäisiä sulkuja, olemme Pythonin rivityssääntöjen puitteissa hieman siistineet koodin esitysasua jakamalla funktiokutsun useammalle riville. Muuten ohjelma on kääntynyt kuten aikaisemmassakin esimerkissä. Meillä on edelleen kansiot *build* ja *dist*, joista *build* sisältää enää prosessinaikaisia tiedostoja, jotka voidaan poistaa, ja *dist* varsinaisen kootun ohjelman.

Huomioita paketoijasta

Paketoijan käyttäminen tällä erää helpoin tapa jakaa Python-ohjelmia sellaisten ihmisten käytettäväksi, jotka eivät halua tai pysty asentamaan Python-kehitysympäristöä tietokoneelleen. Lisäksi itsenäiseksi kokonaisuudeksi paketoidut ohjelmat toimivat tavallisesti hieman nopeammin kuin suoraan tulkin avulla suoritetut. Tämä ei kuitenkaan tarkoita sitä, että halutessasi jakaa tekemäsi Python-ohjelma muiden ihmisten kanssa paketoitu ohjelma olisi aina paras vaihtoehto, koska tilanteeseen liittyy muutama seikka, joihin tulee kiinnittää huomiota.

Ensinnäkin, paketoitu lähdekoodi on ”lukittu” versio ohjelmasta. Tämä tarkoittaa sitä, että et pysty enää jälkeenpäin muuttamaan ohjelman toimintaa koska pakettiin sisällytetään ainoastaan tulkin luomat esikäännetyt versiot lähdekoodista. Tällöin ohjelmastasi löytyviä virheitä ei myöskään voida enää korjata käyttäjien toimesta. Tämän vuoksi paketoituina jaettujen ohjelmien tuleekin olla erityisen hyvin testattuja ja tarkastettuja virheiden varalta.

Toinen ongelma on tilantarve. Kuten aiemmin mainitsimme, paketoi py2exe lähdekoodin mukaan työkalut, joiden avulla ohjelma voidaan ajaa ilman erillistä Python-tulkkia. Tietenkin tämä tarkoittaa myös sitä, että itsenäisen toiminnan varmistamista varten paketoijan on sisällytettävä kaikki toiminnalliset osat tulkista sekä käytetyistä standardikirjastoista mukaan kokonaisuuteen ja tämä vie tilaa. Jos esimerkiksi tarkastelemme äsken luomaamme ohjelmaa, voimme havainnollistaa numeroina tämän ongelma. Lähdekooditiedoston koko on 1,19 kilotavua, kun taas paketoidun ohjelman - käytännössä *dist*-kansion - koko on 8,10 megatavua (8100 kilotavua). Käytännössä ohjelman koko siis kasvoi 8000-kertaiseksi. Tämä johtuu näet siitä, että huolimatta paketoitavan lähdekoodin koosta, joutuu py2exe paketoimaan ohjelman mukaan kahdeksan megatavun verran ajamiseen tarvittavia tiedostoja. Itse lähdekoodien osuus paketin koosta onkin hyvin vähäinen; kaikki pakettiin liitetyt esikäännetyt lähdekoodit ovat library.zip-nimisessä pakatussa tiedostossa.

Numeroina tämä voidaan esittää myös toisella tapaa: kaikki tämän oppaan ohjelmointiesimerkit, joita on yhteensä 9 kappaletta, vievät lähdekooditiedostoina yhteensä tilaa n. 10 kilotavua. Jos loisimme jokaisesta ohjelmasta itsenäisen paketin, veisivät paketit 9*8 eli 72 megatavua kiintolevytilaa. Jos toimittaisin oppaan esimerkkejä esimerkiksi oikolukijoille sähköpostilla tarkasteltavaksi, ei heistä varmastikaan kukaan olisi ilahtunut 70 megatavun sähköpostiliitteestä, joka luultavasti tukkisi useimmat sähköpostilaatikot.

Mikäli haluat lukea lisää py2exe-paketoijan käytöstä ja toiminnasta, on py2exe-projektin kotisivuilla [10] tutoriaaleja sekä lisää esimerkkejä sen käytöstä. Sivut ovat englanninkieliset.