Python-ohjelmointiharjoituksia

- Säädä käyttämäsi ohjelmaeditori sellaiseksi että se ei laita tabulointimerkkejä tiedostoon vaan pistää sinne 3 välilyöntimerkkiä silloin kun tabulointinäppäintä painetaan. Tämä on erittäin tärkeää Python-ohjelmoinnissa koska if-rakenteet ja silmukat tehdään sisentämällä. Valmiissa ohjelmissa sisennykset on tehty 3:lla välilyönnillä. Jos nyt sekaan heitettäisiin tabulointimerkkejä, ohjelmien logiikka voi muuttua hyvin arvaamattomaksi.
- Yleisestikin on niin että jos sisennykset aina tehdään välilyöntimerkeillä tabulointimerkkien sijaan, ohjelmat näyttävät aina samoilta riippumatta siitä millä editorilla tai työkalulla niitä katsellaan. Tämä koskee muitakin kuin Python-ohjelmia.

Kari Laitinen
http://www.naturalprogramming.com
2009-01-06 Tiedosto luotu.
2014-01-12 Viimeisin muutos

TÄSSÄ MUUTAMIA FAKTOJA PYTHON-STRINGEIHIN LIITTYEN:

- Ns. stringiliteraalit voidaan muodostaa joko yksinkertaisia tai kaksinkertaisia heittomerkkejä käyttäen. Siis ’joopa joo’ ja "joopa joo" tarkoittaa Pythonissa samaa asiaa. Kannattaa kuitenkin käyttää kaksinkertaisia eli tuplaheittomerkkejä kuten on totuttu käyttämään muissa ohjelmointikielissä.
- Pythonissa ei ole varsinaista merkkityyppiä kuten esim. char. Pythonia käytettäessä voidaan ajatella että yksimerkkiset stringit ovat näitä "char"-olioita.
- Python-ohjelmointikielessä, samoin kuin esim. Javassa, merkkijonot eli stringit ovat muuttumattomia olioita eli niitä ei voi modifioida sen jälkeen kun olio on luotu. Kun stringiä halutaan muuttaa, on luotava uusi olio vanhan olion pohjalta.
- Stringi voidaan muuttaa listaksi list()-standardifunktion avulla ja lista voidaan muuttaa stringiksi join()-funktion avulla. Näitä on käytetty esim. ohjelmassa Elvis.py. Näitä muunnoksia voidaan tarvita stringejä modifioitaessa. Toisin kuin stringi, lista on tietorakenne jota voidaan vapaasti modifioida.

HARJOITUKSIA OHJELMALLA GuessAWord.py

Seuraavasta kansiosta löytyy Python-ohjelma nimeltä **GuessAWord.py**
`http://www.naturalprogramming.com/pythonprograms/pythonfilesextra/`
Tämä ohjelma on yksinkertainen tietokonepeli, jossa pelaajan täytyy yrittää arvata ohjelman 'tietämän' sanan kirjaimia. Pelin edistyessä pelaaja voi yrittää arvata myös koko sanan. Tutki ohjelmaa ja tee sitten seuraavia harjoituksia.

Harjoitus 1:

Paranna ohjelmaa siten että arvattava sana otetaan ohjelman käynnistyessä satunnaisesti listasta joka sisältää arvattavia sanoja. Tällainen lista voidaan muodostaa esim. seuraavanlaisella lauseella:

```
words_to_be_guessed = [ "VIENNA", "HELSINKI", "COPENHAGEN",
                        "LONDON", "BERLIN", "AMSTERDAM" ]
```

Satunnainen indeksi yllä annettua listaa varten voidaan tehdä `random.random()` metodilla seuraavaan tapaan

```
int ( ( random.random() * len( words_to_be_guessed ) ) )
```

Metodi `random.random()` palautaa `double`-arvon alueelta 0.0 .... 1.0 siten että arvoa 1.0 ei koskaan palauteta. Metodilla (funktiolla) `len()` saadaan selville listan pituus ja metodilla `int()` voidaan laskettu `double`-arvo pyöristää (alaspäin) `int`-arvoksi. Metodin `random.random()` käyttämiseksi tarvitset ohjelman alkuu seuraavanlaisen `import`-lauseen:

```
import random
```

Esimerkki `random.random()` -metodin käytöstä löytyy ohjelmasta **MathDemo.py**.

Harjoitus 2:

Nyt ohjelma on sellainen että se päättyy silloin kun peli päättyy. Muuta ohjelma sellaiseksi että peliä voidaan pelata useaan kertaan yhden ohjelman suorituskerran aikana. Edellä mainitusta kansiosta löytyy ohjelma nimeltä **RepeatableGame.py** josta voi katsoa mallia tällaisen ominaisuuden toteutukseen.

Harjoitus 3:

Paranna ohjelmaa siten että se laskee kuinka monta arvausta pelaaja tekee yhden pelin aikana. Pelin pelaamisen jälkeen ja ennen uuden pelin aloitusta ohjelman tulee tulostaa kuinka monta arvausta tehtiin. Seuraavanlaista muuttujaa voidaan käyttää arvausten määrän laskentaan:

```
number_of_guesses = 0
```

Harjoitus 4:

Paranna ohjelmaa siten että se tulostaa tiedot pelatuista peleistä ennenkuin ohjalma päättyy. Tällainen ’pelistatistiikka’ voi näytää esim. seuraavanlaiselta:

```
PLAYED WORD      GUESSES

COPENHAGEN         7
LONDON             6
COPENHAGEN         4
```

```
BERLIN              5
HELSINKI            4
```

Yllä esitetyssä ensimmäisen pelin arvattava sana oli COPENHAGEN ja sitä pelaaja arvasi 7 kertaa, toisen pelin sana oli LONDON ja pelaaja arvasi sitä 6 kertaa, jne. Koska arvattavat sanat arvotaan satunnaisesti taulukosta, on mahdollista että jotkut sanat tulevat pelatuiksi useaan kertaan.

Seuraavanlaisia tietorakenteita voi käyttää pelattujen pelien tietojen talletukseen.

```
games_played = 0
played_words = []
guesses_in_games = []
```

Muuttujalla `games_played` voidaan laskea pelattujen pelien määrää, ja listarakenteisiin voidaan tallettaa pelatut sanat ja niihin liittyvät arvausten määrät. Uusi alkio voidaan lisätä listan perään metodilla nimeltä `append()`. Jokaisen pelin lopussa voidaan listarakenteisiin lisätä kyseisen pelin tiedot, ja nämä tiedot voidaan sitten tulostaa lopuksi kun käyttäjä ei enää halua pelata pelejä.

HARJOITUKSIA OHJELMALLA Animals.py

Harjoitus 1:

Lisää luokkaan `Animal` metodi `make_stomach_empty()`, jonka avulla `Animal`-olion vatsa tyhjennetään siten että sinne kirjoitetaan tyhjä stringi "". Metodia tulee voida kutsua esim. seuraavasti

```
cat_object.make_stomach_empty()
dog_object.make_stomach_empty()
```

Tämän muutoksen onnistumisen voi testata kutsumalla `make_speak()` -metodia ja tutkimalla onko vatsa todella tyhjentynyt.

Harjoitus 2:

Muuta `Animal`-luokan konstruktoria siten että sen parametrille tulee oletusarvo. `Animal`-tyypin olio voidaan luoda tämän seurauksena parametreja antamatta seuraavasti

```
default_animal  =  Animal()
```

Datakentän `species_name` arvoksi voidaan oletusarvoisesti laittaa stringi "default animal".

Esim. ohjelmasta **Windows.py** näet kuinka konstruktorille asetetaan parametrien oletusarvot. Tämänkin tehtävän onnistumisen voit todeta `make_speak()` -metodin avulla.

Harjoitus 3:

Python-olioiden datakenttiä eli datajäseniä kutsutaan myös attribuuteiksi tai tarkemmin sanoen instanssiattribuuteiksi. Python-luokissa ei olioiden datakenttiä luetella (deklaroida) erikseen vaan datakentät syntyvät kun jonkin metodin sisällä kirjoitetaan

```
self.data_field_name = ...
```

Lisää luokkaan `Animal` uusi datakenttä (instanssiattribuutti) kirjoittamalla konstruktoriin

```
self.animal_name = ...
```

Tässä on muutettava luokan konstruktoria siten että `Animal`-olio voidaan luoda esim. lauseella

```
named_cat  =  Animal( "cat", "Arnold" )
```

Myöskin olion kopiointia on muutettava siten että uusi datakenttä tulee kopioiduksi. Metodia `make_speak()` tulee modifioida siten että se tekee seuraavantapaisen tulostuksen

```
Hello, I am a cat named Arnold.
I have eaten: ...
```

Uudelle datakentälle voidaan antaa oletusarvoksi "nameless".

Harjoitus 4:

Muuta metodia `make_speak()` siten että se tulostaa

```
Hello, I am a ... named ...
My stomach is empty.
```

siinä tapauksessa kun `stomach_contents` viittaa tyhjään stringiin. Vatsa on tyhjä niin kauan kuin metodia `feed()` ei ole kutsutta. Voit käyttää standardifunktiota `len()` tarkistamaan onko vatsa tyhjä. Funktiota `len()` voi käyttää esimerkiksi seuraavaan tapaan

```
if len( self.stomach_contents )  ==  0 :

   # stomach_contents viittaa tyhjaan stringiin.
   ...
```

Jos vatsa ei ole tyhjä, annetaan alkuperäisen kaltainen tulostus.

Harjoitus 5:

Muuta `feed()`-metodia siten että tällä uudella metodilla voi syöttää myös toisen eläimen jollekin `Animal`-oliolle. Metodi `feed()` voi toimia samaan tapaan kuin luokan konstruktori toimii, eli siinä tutkitaan minkä tyyppinen parametri on annettu. Jos on annettu `Animal`-tyyppinen parametri, syödään sitten se `Animal`-olio.

Toisen eläimen syöminen voi tapahtua esimerkiksi siten että syötävän eläinparan datajäsen `animal_name` kulkeutuu sen syövän `Animal`-olion vatsaan. Uudessa `feed()` -metodissa voidaan viitata argumenttina (parametrina) tulevan `Animal`-olion datajäseneen `animal_name` samaan tapaan kuin konstruktorissakin viitataan parametrina saadun olion datajäseniin

Syödyn eläimen datakenttä `animal_name` voi muuttua syömisoperaatiossa nimelle "Eaten animal". Kun uusi `feed()`-metodi on olemassa pitäisi lauseiden

```
tiger_object  =  Animal( "tiger", "Richard" )
cow_object    =  Animal( "cow", "Bertha" )

tiger_object.feed( cow_object )
tiger_object.make_speak()
```

tuottaa tulostus

```
Hello, I am a tiger named Richard
I have eaten: Bertha,
```

Harjoitus 6:

Muuta `Animal`-luokan datajäsen `stomach_contents` string-olioita sisältäväksi listaksi joka määritellään konstruktorissa seuraavaan tapaan

```
self.stomach_contents  =  []
```

Tarkoitus on että eläinoliota ruokittaessa ruokana annettava stringi lisätään tämän listan loppuun.

Tämä muutos vaatii muutoksia konstruktoriin ja muihin luokan metodeihin. Metodeiden "signeerauksia", siis niiden ottamia parametreja, ei tarvitse muuttaa. Näin myöskään "pääohjelmaa" ei tarvitse tässä kohdassa muuttaa.

Esimerkiksi metodissa `feed()` tulee tehdä sellainen muutos että annettu ruokastringi lisätään `append()`-metodilla ruokalistan loppuun.

Vatsan sisällön tulostaminen tarkoittaa sitten ruokastringien tulostamista listasta. Yksinkertaisimmillaan listan stringit voidaan tulostaa esim. seuraavasti:

```
list_of_strings  =  [ "first", "second", "third" ]

for string_to_print in list_of_strings :

   print string_to_print
```


HARJOITUKSIA ISODate-LUOKALLA

Tiedosto ISODate.py sisältää ISODate-luokan jonka avulla voidaan tehdä erilaisia päivämääriin liittyviä laskutoimituksia. Tätä kyseistä luokkaa käyttävät esimerkiksi ohjelmat Columbus.py, Birthdays.py ja Friday13.py.

Luokka ISODate käsittelee päivämääriä ns. ISO-formaatissa, mikä tarkoittaa että päivämäärät syötetään luokkaan ja tulostetaan järjestyksessä VVVV-KK-PP

Harjoitus 1:

Tee ohjelma, joka laskee ISODate-luokan avulla nykyisen ikäsi vuosissa, kuukausissa ja päivissä. Tämän voit tehdä kun määrittelet ohjelmaasi ISODate-olioita seuraavaan tapaan

```
my_birhtday =  ISODate( 1977, 7, 14 )
date_now    =  ISODate()
```

Ohjelmasta Columbus.py näet kuinka kahden ISODate-olion ajallinen etäisyys voidaan laskea. Voit myös helposti ohjelman avulla ottaa selville minä viikonpäivänä olet syntynyt. Jos otat pohjaksi Columbus.py-ohjelman, muuta ohjelmassa käytetyt nimet sellaisiksi että ne ovat kuten esimerkiksi yllä.

Harjoitus 2:

Kopioi sopivasti koodia ohjelmasta Birthdays.py siten että nyt tekeillä oleva ohjelma tulostaa taulukon jossa kerrotaan milloin on tärkeimmät syntymäpäiväsi ja mille viikonpäiville ne sattuvat. Joudut luonnollisesti muuttamaan nimiä siten että saat Birthdays.py-ohjelmassa olevan silmukan toimimaan omassa ohjelmassasi.

Harjoitus 3:

Pythonissa luokat voivat periä toisia luokkia. Esimerkkejä joissa perintää on käytetty ovat mm. BankPolymorphic.py ja Windows.py.

Johda eli periytä luokasta ISODate uusi luokka nimeltä AnotherDate, jonka tulee olla muuten samanlainen kuin ISODate-luokka mutta siinä tulee olla lisäksi metodi joka alkaa seuraavasti

```
def to_anti_iso_format( self )  :
    # tästä alkaa sitten metodin sisäiset lauseet
```

Tarkoitus on että tämä metodi palauttaa stringin joka sisältää päivämäärän anti-ISO -formaatissa joka on sitten PP.KK.VVVV. Saat tämän metodin helposti aikaiseksi kun otat kopion metodista __str__(self) ja muutat sen nimen ja vähän sisältäkin sopivasti. Tällä metodilla voidaan sitten päivämäärät tulostaa yleisesti käytetyssä muodossa seuraavaan tapaan

```
print "\n\n   " +  test_date.to_anti_iso_format()
```

Huomaa että kun periytät uuden luokan niin Python-kielessä konstruktoriksi sanottu __init__(self) -metodi periytyy myös. Näin siis alaluokkaan ei aina tarvitse kirjoittaa uuttaa

konstruktoria.
Voit kirjoittaa uuden luokan ohjelmatiedostosi alkuun import-lauseiden jälkeen.

Harjoitus 4:

Lisää ohjelmaan ominaisuus, että se tulostaa päivämäärät jolloin olet 10000 ja 20000 päivää vanha. Ihmisen ikä on 10000 päivää kun hänen ikänsä on suurinpiirtein 27 vuotta ja 4 1/2 kuukautta. Tämän ominaisuuden avulla saat sitten uusia juhlimispäiviä normaalien syntymäpäivien lisäksi. Tämän ominaisuuden saat aikaiseksi esim. kun kasvatat silmukassa päivälaskuria ja samalla inkrementoit ISODate-oliota seuraavaan tapaan

```
day_counter         =  0
date_to_increment   =  ISODate( my_birthday )

while  ... :

   day_counter += 1
   date_to_increment.increment()

   if ...
```

Harjoitus 5:

Lisää ohjelmaan ominaisuus että se ilmoittaa milloin olet miljardi sekuntia, siis 1000000000 s, vanha. Tämän saat tehtyä myös siten että lasket päiviä syntymäpäivästäsi lähtien. Vuorokaudessa on 24 * 60 * 60 sekuntia. Miljardi sekuntia saavutetaan joskun 31 ikävuoden jälkeen. Ohjelmasi voi lisäksi ilmoittaa tarkan ikäsi vuosina, kuukausina, ja päivinä silloin kun olet miljardi sekuntia vanha.

Ensimmäinen PyQt-harjoitus: Shakkilaudan piirtäminen

Näissä harjoituksissa opettelemme PyQt-kirjaston piirtometodien käyttöä. Harjoitusten pohjaksi voit ottaa esim. HelloQt.py -ohjelman.

Harjoitus 1:

Katso mallia ohjelmasta DrawingDemoQt.py ja opettele piirtämään piirtoalueelle neliö.

Harjoitus 2:

Kun osaat piirtää neliön, tee ohjelmasta sellainen että se piirtää näkyville shakkilaudan jossa on 8 x 8 ruutua joista joka toinen ruutu on musta ja joka toinen valkoinen. Helpoiten shakkilauta syntynee kun ohjelmaan pistää silmukat jotka piirtävät shakkilaudan ruudut.

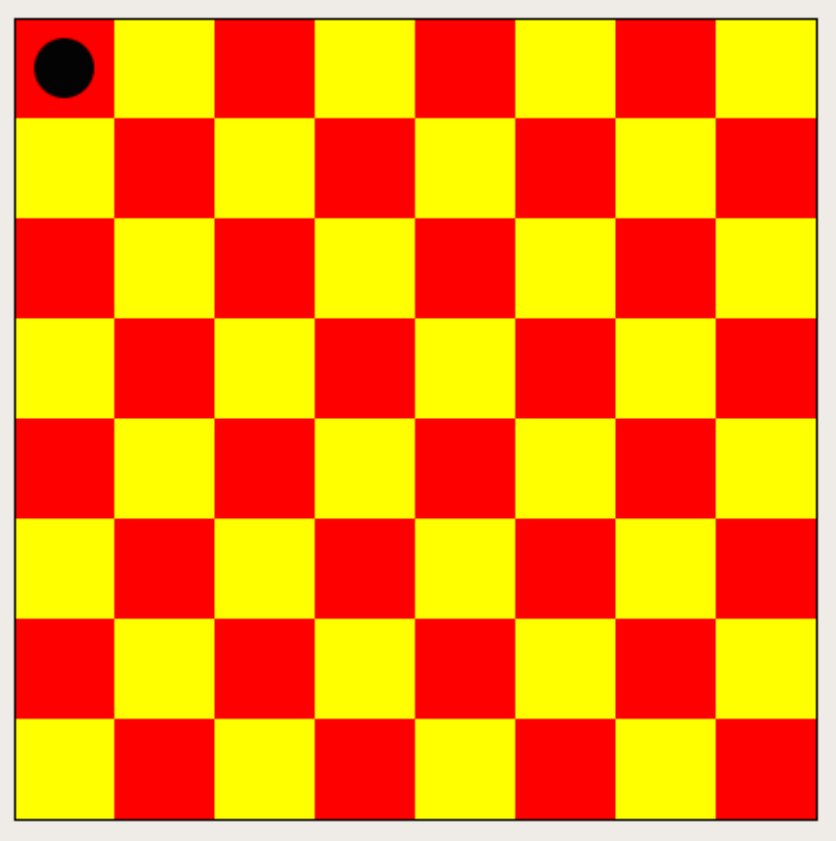
Valkoiset ruudut voi aluksi jättää piirtämättä ja antaa niiden olla taustan värisiä. Shakkilaudan ympärille voi piirtää vielä raamit.

Harjoitus 3:

Tee shakkilaudan "mustista" ruuduista punaisia ja "valkoisista" ruuduista keltaisia. Tässä kannattaa "valkoiset" ruudut tehdä siten että piirtää ensin taustalle ison keltaisen neliön ja tuon neliön päälle sitten piirretään "mustat" punaiset ruudut.

Harjoitus 4:

Piirrä vasempaan yläkulmaan ruutuun "shakkinappulaksi" pallonmuotoinen nappula mustalla värillä. Tässä vaiheessa shakkilautasi voisi näyttää esim. seuraavanlaiselta



Harjoitus 5:

Komentorivi-ikkunassa suoritettavien ohjelmien joukosta löydät ohjelman nimeltä MathDemo.py jossa mm. kerrotaan yksi tapa satunnaislukujen generointiin. Käytä hyväksi satunnaislukuja ja pistä edellisessä kohdassa piirretty nappula johonkin satunnaiseen shakkilaudan ruutuun. Tarkoitus on että aina kun ikkuna "maalataan" shakkinappula näkyy jossain satunnaisessa paikassa ruudulla.

Harjoitus 6:

Tutki PyQt-kirjaston dokumentaatiota ja ota selville miten piiroväri voidaan tehdä numeeristen RGB-arvojen avulla. Kun osaat tämän, tee shakkilaudasta sellainen että sen kaksi väriä, jotka ovat aiemmin olleet keltainen ja punainen, ovat satunnaisesti arvottuja värejä.

HARJOITUKSIA OHJELMALLA AnimationDemoQt.py

Harjoitus 1:

Nykyisellään ohjelmassa vilkkuva pallo pysyy paikallaan suurinpiirtein ikkunan keskellä. Muuta ohjelmaa siten että pallo alkaa hiljalleen liikkua alaspäin. Siis kun pallo tulee seuraavan kerran näkyviin, se näkyy muutamaa pikseliä alempana. Tässä pitää myös vartioida että pallon paikan y-koorinaatti ei kasva niin suureksi että pallo katoa kokonaan ruudulta. Tämän tehtävän ratkaisun seurauksena pallo jää vilkkumaan ikkunan alareunaan sitten kun alareuna on saavutettu.

Pallon y-koordinaatti voidaan kätevästi asettaa valmiiksi seuraavaa piirtokertaa varten paintEvent()-metodissa sen jälkeen kun pallo on jo piirretty nykyisillä koordinaateilla.

Harjoitus 2:

Paranna edellisessä kohdassa tehtyä ominaisuutta siten että pallo lähtee nousemaan hiljalleen ylöspäin sitten kun se on saavuttanut ikkunan alareunan. Kun pallo on saavuttanut ikkunan yläreunan, se lähtee taas hiljalleen valumaan alaspäin. Tarkoitus on että pallo lopulta "sahaa" jatkuvasti ikkunan ala- ja yläreunan välissä. Ohjelmaan kannattaa lisätä seuraava datajäsen (instanssiattribuutti) jolla pallon liikettä voiaan kontrolloida:

```
self.moving_down  =  True
```

Tälle muuttujalle voidaan antaa arvo False sitten kun palloa aletaan siirtämään ylöspäin.

Harjoitus 3:

Muuta ohjelma sellaiseksi että se ei liikutakaan palloa vaan kuvaa. Ohjelmasta SinglePictureQt.py näet kuinka kuva saadaan piirrettyä QImage-oliona. Kuvaksi kannattaa ottaa seuraavia osatehtäviä silmälläpitäen jokin neliönmuotoinen nuolen kuva. Nuolikuvia voit löytää esim. kansiosta http://www.oamk.fi/~karil/images/arrow_images/

Harjoitus 4:

Jos laitoit edellisessä osatehtävässä kuvaksi nuoli alas -kuvan, muuta ohjelma sellaiseksi että nuoli osoittaa alaspäin silloin kun kuva liikkuu alaspäin, ja nuoli muuttuu ylöspäin osoittavaksi nuoleksi silloin kun liikutaan ylöspäin. Voit tässä käyttää kahta erillistä nuolikuvaa, mutta homma hoituu kätevästi myös QImage-luokasta löytyvän metodin avulla jolla kuva voidaan muuttaa itsensä peilikuvaksi.

Harjoitus 5: (Tämän onnistumisesta ei ole takeita QImage-luokalla.)

Jos innostusta piisaa, muuta ohjelma sellaiseksi että nuolikuva liikkuu ympäri ikkunan reunaa ja nuoli osoittaa aina menosuuntaan.

HARJOITUKSIA OHJELMALLA MouseDemoQt.py

Tässä harjoituksessa voit ottaa pohjaksi ohjelman MouseDemoQt.py josta näet kuinka hiiren nappien painalluksiin, hiiren liikutteluun ja hiiren nappien vapauttamiseen reagoidaan PyQt-kirjastoon perustuvissa GUI-ohjelmissa.

Harjoitus 1:

Poista ohjelmasta sen nykyisellään aiheuttamat tulostukset ja tee siitä sellainen että piirtoalueelle piirretään jokin kuva heti kun ohjelma käynnistyy. Ohjelmasta SinglePictureQt.py näet kuinka kuva saadaan näytettyä QImage-oliona.

Harjoitus 2:

Kun olet saanut kuvan näkyville, paranna ohjelmaa siten että hiirellä voi "tarttua" kuvaan ja liikuttaa sitä eri kohtaan ruudulla. Tarkoitus on että kuvaa voidaan liikuttaa nimenomaan hiiren vasemmalla napilla siten että

- kun hiiren vasen nappi painetaan alas siten että kursori on kuvan päällä, aloitetaan kuvan siirto-operaatio, ja otetaan talteen paikka josta kuvaa alettiin siirtää
- kun hiirtä liikutetaan, on laskettava paljonko hiiri on liikkunut, ja siirrettävä kuvaa vastaava määrä
- kun hiiren nappi vapautetaan, kuvan siirto-operaatio päättyy, ja kuva jää sille paikalle.

Tässä saattaa olla hyvä käyttää boolean-tyyppistä datajäsentä (instanssiattribuuttia)

```
self.picture_is_being_moved  =  False
```

joka asetetaan arvoon True siksi ajaksi jolloin kuvaa siirretään. Tämmöisen muuttujan avulla voidaan hallita tilanne jossa hiiren nappi painetaan alas kuvan ulkopuolella ja vapautetaan kuvan päällä, jolloinka kuvaa ei pidä siirtää.

Sen tutkiminen että onko klikkauspiste kuvan päällä voidaan suorittaa seuraavan tapaisella ohjelmakoodilla:

```
def mousePressEvent( self, event ) :

   current_mouse_position_x = event.x()
   current_mouse_position_y = event.y()

   if current_mouse_position_x > self.picture_position_x and \
      current_mouse_position_x < self.picture_position_x + self.picture_width and \
      current_mouse_position_y > self.picture_position_y and \
      current_mouse_position_y < self.picture_position_y + self.picture_height :

      #  The picture area was clicked.
```

Harjoitus 3:

Muuta kuva näkymään tummempana sinä aikana kun sitä siirretään. Tämmöinen pieni juttu saattaa tehdä ohjelman käyttäjäkokemuksesta vaikuttavamman.

Tämmöinen ominaisuus saadaan aikaiseksi kun kuvan päälle piirretään suorakaide hyvin läpäisevällä mustalla värilla silloin kun kuvaa ollaa siirtämässä.

Harjoitus 4:

Muuta ohjelmaa siten että hiiren oikealla napilla voidaan näytettävän kuvan kokoa muuttaa. Tee ohjelmasta sellainen että jos painetaan hiiren oikeanpuoleista nappia niin näytettävän kuvan koko kasvaa hieman. Jos tuota oikeata nappia painetaan ja Ctrl-näppäin on samanaikaisesti alas painettuja, tulee näytettävän kuvan koon pienentyä hiukan. QImage-luokasta löytyy valmis metodi jolla kuvan kokoa voidaan muuttaa siten että olemassa olevasta kuvasta tehdään uusi QImage-olio jossa kuva on pienempänä tai suurempana.

HARJOITUKSIA OHJELMALLA MovingBallQt.py

Ota pohjaksi opettajan neuvojen mukaisesti joko ohjelma MovingBallQt.py tai ohjelma MovingBallOOQt.py. Jälkimmäinen on oliosuuntautuneesti rakennettu versio samasta ohjelmasta, jota käyttäessäsi tarvitset lisäksi tiedoston BallQt.py.

Harjoitus 1:

Lisää ohjelmaan uusi painonappi joka toimii Reset-nappina siten että sitä painettaessa pallon paikaksi tulee sen paikka ohjelman käynnistyessä ja pallon väriksi tulee sillä alussa ollut väri.

Ohjelmaan tarvitaan tässä uusi QPushButton olio. Lisää ohjelmaan ensin tämä uusi painonappiolio ja tarkista että se ilmestyy sovelluksen ikkunaan. Lisää vasta tämän jälkeen nappiin siihen vaadittu toiminto.

Harjoitus 2:

Lisää ohjelmaan ominaisuus että pallon kokoa voi säätää ns. sliderilla joka on PyQt:ssa esim. luokan QSlider olio. Katso mallia ohjelmasta CurtainsQt.py jossa näitä slidereita on käytetty.

Luokkaan voidaan tarvita, mikäli pohjana on ohjelma MovingBallQt.py, datajäseneksi pallon kokoa kuvaava muuttuja jonka arvon tulee muuttua sliderin avulla. Muuttuja voi olla esim.

```
self.ball_diameter  =  100
```

Harjoitus 3:

Muuta ohjelma sellaiseksi että pallon ei sallita menevän ikkunan ulkopuolelle. Tutki QPushButton-luokan dokumentaatiota ja ota selville kuinka painonappi deaktivoidaan silloin kun pallo on saavuttanut apletin alueen reunan. Painonappi tulee saattaa takaisin aktiiviseksi sitten kun palloa on liikutettu takaisin apletin keskikohtaa päin.

Harjoitus 4:

Muuta pallon värinvalinta sellaiseksi että värin voi valita QColorDialog-luokan avulla. Tarvitset luultavimmin painonapin jolla kyseinen dialogi käynnistetään.

Harjoitus 5 (tehdään vain jos oliosuuntautunut MovingBallOOQt.py käytössä):

Muuta ohjelma sellaiseksi että alussa ruudulla näkyy kolme palloa rinnakkain, ja ainoastaaan yksi palloista on aktiivinen eli operaatiopaneelin säätimet vaikuttavat palloon. Aktiivisen pallon valitsemiseksi ohjelmaan on lisättävä nappi jolla aktiivinen pallo voidaan vaihtaa.

Ball-olioita voi luoda esim. konstruktorin lopussa:

```
self.eka_pallo  = Ball( ...
self.toka_pallo  =  Ball( ...
self.kolmas_pallo  =  Ball( ...

self.aktiivinen_pallo  =  self.eka_pallo
```

Kun käytetään erillistä aktiivinen_pallo -olionimeä, voidaan tämä nimi panna viittaamaan aina siihen Ball-olioon joka on aktiivinen. Tätä nimeä voidaan käyttää palloon vaikuttavissa metodeissa ja näin viitataan automaattisesti siihen Ball-olioon joka on kulloinkin aktiivinen. Kun aktiivinen pallo halutaan vaihtaa toiseksi, se voidaan tehdä esimerkiksi seuraavanlaisella rakenteella

```
....
   if  self.aktiivinen_pallo  ==  self.eka_pallo :

      self.aktiivinen_pallo  =  self.toka_pallo

   elif  self.aktiivinen_pallo  ==  self.toka_pallo :

      aktiivinen_pallo  =  self.kolmas_pallo

   elif  self.aktiivinen_pallo  ==  self.kolmas_pallo :

      self.aktiivinen_pallo  =  self.eka_pallo
```

Harjoituksia menujen käyttöön liittyen PyQt-ohjelmissa

Ota näiden harjoitusten pohjaksi käyttöön ohjelma ClockQt.py.

Harjoitus 1:

Lisää ClockQt.py-ohjelmaan File- ja Settings-menut. Tässä voit ottaa mallia ohjelmasta MenuDemoApplicationQt.py.

Menuja rakennettaessa kannattaa edetä siten että ensin testataan että menut tulevat näkyviin ja vasta sitten menuihin pannaan toimintaa.

Laita File-menuun toiminto File -> Exit jolla ohjelmasta voidaan poistua.

Laita Settings-menuun valinta Settings ->Clock Background Color, josta pääsee toisensa poissulkeviin valintoihin Black ja White. Näillä on mahdollista valita kellon taustaksi joko musta tai valkoinen tausta.

Kellon piirtävässä metodissa pitää sitten piirtää mustalle kellolle musta tausta ja itse kello pitää piirtää valkoisella. Valkoiselle kellolle tehdään valkoinen tausta ja kello piirretään mustana.

Harjoitus 2:

Tee kellosta herätyskello siten että herätysaika voidaan antaa Settings-menun kautta. Menussa pitää olla erikseen toiminnot herätyksen aktivoimiseksi ja herätysajan asettamiseksi. Ohjelma voi toimia niin että herätys aktivoidaan automaattisesti aina kun herätysaikaa muutetaan. Toisaalta pitää olla mahdollisuus herätyksen deaktivointiin.

Kellon käydessä pitää sitten antaa hälytys jos kellon aika saavuttaa asetetun herätysajan. Hälytys voi olla jokin visuaalinen juttu kuten esim. että näytölle kirjoitetaan teksti "HERÄÄ JO!", tai että kellon tausta alkaa vilkkua eri väreillä.

Herätyskello pitää sitten saada "hiljaiseksi" kun hälytys on "kuultu". Tämä voi tapahtua esimerkiksi siten että kello hiljenee kun jotain näppäintä painetaan tai hiirellä klikataan.

Harjoituksia ohjelmalla FlagsQt.py

Ohjelma FlagsQt.py on esimerkki suurehkosta oliosuuntautuneesti rakennetusta PyQt-ohjelmasta. Siinä näytetään kuinka niinsanottuihin välilehtiin perustuva käyttöliittymä rakennetaan QTabWidget -luokan avulla.

Harjoitus 1:

Lisää johonkin ohjelman välilehdistä uusi jonkin kuvitteellisen maan lippu. Esim. Absurdistanin lipussa voisi olla pystysuorina eli vertikaalisina raitoina kaikki sateenkaaren värit. Tämmöisen lipun voi kätevästi tehdä VerticalStripesFlag -luokan avaulla.

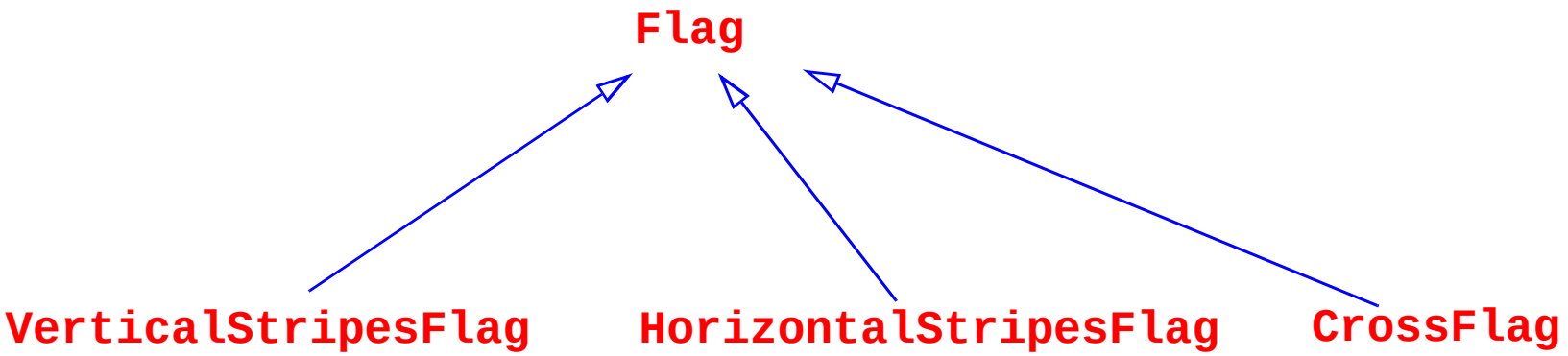
Harjoitus 2:

Lisää ohjelmaan uusi välilehti "Fictitious" johon lisäät edellisessä tehtävässä tekemäsi kuvitteellisen lipun. Tämä välilehti tehdään samaan tapaan kuin muutkin välilehdet eli luodaan uusi FlagPanel-olio johon uusi lippu lisätään.

On mahdollista että ohjelma ei toimi jos FlagPanel-olioon ei ole lisätty yhtään lippua.

Harjoitus 3:

Ohjelmassa FlagsQt.py luokka Flag toimii yläluokkana useille alaluokille jotka edustavat eri tyyppisiä lippuja. Luokkien hierarkia on seuraavanlainen



Jokaisella alaluokalla on oma draw()-metodi joka kykenee piirtämään kyseisen luokan edustaman lipun. Oikea draw()-metodi tulee valituksi automaattisesti kun lippuoliot on listaan talletettuna.

Johda Flag-luokasta uusi luokka joka edustaa sellaisia lippuja joiden keskellä on pallonmuotoinen kuvio. Esim. Japanin lipussa on punainen pallo (keskellä) valkealla pohjalla. Bangladeshin lipussa on punertava pallo vihereällä pohjalla. Uuden luokan nimi voi olla BallFlag ja sen draw()-metodi piirtää siis automaattisesti määritellynvärisen pallon keskelle lippua. (Voit käyttää Bangladeshin lippua testilippuna, vaikka sen pallo ei taida virallisesti sattuakaan ihan keskelle.)

Tämän uuden luokan testaamiseksi tulee jollekin välilehdelle lisätä BallFlag-lippu(ja).

Harjoitus 4:

Johda Flag-luokasta uusi luokka nimeltä SingleStarFlag. Tuon luokan liput ovat sellaisia että niissä on yksi viisihaarainen tähti keskellä lippua. Tämän tyyppinen lippu on esim. Vietnamilla, Somalialla ja Marokolla, vaikka tähti ei olekaan kaikissa näissä lipuissa ihan samankokoinen.

Tähtikuvion piirtämiseksi voidaan käyttää standardiluokkaa QPainterPath jota on käytetty esim. ohjelmassa StarsQt.py. Toisaalta on niin että ohjelmassa StarsQt.py on valmiina luokka nimeltä StarShape5 joka edustaa viisisakaraista tähteä ja jota voit käyttää ohjelmassasi vaikka et tarkasti tuntisikaan luokan QPainterPath ominaisuuksia. SingleStarFlag-tyyppiseen lippuun tarvittava tähti voidaan määritellä StarShape5-oliona ja se saadaan piirretyksi StarShape5-luokan draw()-metodin avulla. Saat StarShape5-luokan käyttöösi kun ensin kopioit tiedoston StarsQt.py samaan kansioon jossa ohjelmasi on, ja tämän jälkeen käytät ohjelmassasi seuraavaa import-komentoa:

```
from StarsQt import StarShape5
```

Luonnollisesti SingleStarFlag-luokasta on luotava olio jotta voit testata ohjelmasi uutta ominaisuutta.

Harjoitus 5:

(Tämän harjoituksen voit tehdä vaikka harjoituksia 3 ja 4 ei olisikaan tehty.)

Lisää ohjelmaan välilehti jolla voi suunnitella lippuja. Kannattaa aluksi ottaa suunnitteltavaksi lipuksi vain yksi lipputyyppi jossa on esim. 3 väriä.

Tässä pitäisi tehdä erityinen FlagDesignPanel -luokka joka olisi FlagPanel-luokan kaltainen siinä mielessä että paneelissa olisi piirtoalue ja alareunassa jotain säätimiä jolla suunniteltavan lipun värejä voi vaihdella. Tässä voisi edetä esim. siten että kopioi MovingBallQt.py:stä siinä käytetyt napit ja valikon ja tehdä niihin perustuen sellaisen lippujensuunnittelupaneelin että napeilla voi valita mitä lipun raitaa muutetaan ja värivalikosta voisi valita valitulle raidalle uuden värin.

FlagDesignPanel-luokassa on siis vain yksi lippuolio jota muokataan värienvalintasäätimillä. Tässä täytyy olemassaolevan lippuolion värejä muuttaa, joten on mahdollista että Flag-luokkaan tulee tehdä uusia metodeita yksittäisten värien asettamista varten. Toinen mahdollisuus on luoda aina uusi lippuolio kun lipun värejä muutetaan.

Tässä saa käyttää mielikuvitusta! On mahdollista tehdä lippuja myös siten että ohjelma arpoo lipun tyypin ja siinä käytetyt värit satunnaisesti. Jos tekee esim. kolmivärisiä lippuja, voisi tehdä nelinappisen käyttöliittymän joista yhdellä napilla arvotaan lipun tyyppi ja kolmella napilla arvotaan käytetyt värit.

HARJOITUKSIA OHJELMALLA CollidingMiceQt.py

CollidingMiceQt.py on ohjelmaesimerkki jossa esitellään erityisesti seuraavien luokkien käyttöä:

- QGraphicsItem on luokka josta johdettujen luokkien oliot ovat joitain graafisesti piirrettäviä hahmoja. Ohjelmassa CollidingMiceQt.py on luokasta QGraphicsItem johdettu luokka Mouse jonka oliot ovat näytöllä liikkuvia hiiriä. QGraphicsItem-pohjaisilla olioilla on mm. oma koordinaatistonsa.
- QGraphicsScene-luokan avulla voidaan luoda olio joka on eräänlainen maisema jossa QGraphicsItem-luokkaan pohjautuvat oliot voivat oleskella tai liikkua.
- QGraphicsView on luokka jonka avulla voidaan käyttää ja näyttää QGraphicsScene-luokan avulla luotua graafista maisemaa.

Tutki ohjelmaa ja tee sitten seuraavia harjoituksia.

Harjoitus 1:

Johda luokasta Mouse uusi luokka nimeltä BigBlackMouse ja laita tämän luokan olio liikuskelemaan muiden hiirien joukkoon. Yksinkertaisimmillaan saat tämmöisen luokan aikaiseksi kun lisäät seuraavat ohjelmarivit luokan Mouse määrittelyn jälkeen:

```
class BigBlackMouse( Mouse ) :

   def __init__( self ) :

      Mouse.__init__( self )

      self.mouse_color = Qt.black
```

Kun BigBlackMouse-luokka on tehty näin, hiiren väri on kyllä musta mutta hiiri ei ole yhtään suurempi kuin tavallinen Mouse-olio. Seuraavassa harjoituksessa tehdään sitten hiirestä isompi.

Luonnollisesti sinun täytyy tehdä ohjelmaan BigBlackMouse-olio uuden luokan testaamiseksi. BigBlackMouse-oliolle ei välttämättä tarvitse asettaa paikkaa ennenkuin se laitetaan 'grafiikkaskeneen'.

Harjoitus 2:

Tässä on tarkoitus saada BigBlackMouse-oliosta isompi hiiri jotta se olisi nimensä mukainen.

Koska nyt on niin että jokaisella hiirioliolla on periaatteessa oma koordinaatistonsa joka voi olla esim. kääntynyt tai skaalautunut varsinaisen ikkunan koordinaatistoon nähden, on mahdollista tehdä luokan BigBlackMouse-oliolle skaalautunut koordinaatisto siten että musta hiiri näyttää suuremmalta kuin muut hiiret.

Tämän saat aikaiseksi siten että teet luokkaan BigBlackMouse paint()-metodin jossa ensin skaalaat koordinaatistoa suuremmaksi ja sen jälkeen kutsut sieltä yläluokan paint()-metodia seuraavaan tapaan

```
Mouse.paint( self, painter, option, widget )
```

Esim. ohjelmasta FlyingArrowQt.py näet kuinka koordinaatistoa voidaan skaalata. Tässä uuteen metodiin tulee vain pari riviä ohjelmakoodia.

Harjoitus 3:

Normaalisti hiirien korvat muuttuvat punaisiksi silloin kun ne törmäävät johonkin toiseen hiireen. Tämä törmääminen tutkitaan Mouse-luokan paint()-metodissa.

Muuta ohjelma sellaiseksi että jos törmääminen tapahtuu BigBlackMouse-tyyppisen hiiriolion kanssa, "tämän" hiiren korvat maalataan esim. valkoisella.

Tämän homman voi tehdä pelkästään Mouse-luokan paint()-metodia muuttamalla. Siellä käytetään jo nyt listaa joka sisältää törmäävät eli kollideeraavat hiirioliot. Kyseiseen listaan voidaan viitata nimellä seuraavasti:

```
list_of_colliding_mice = \
                    self.scene().collidingItems( self )
```

Yllä olevassa lauseessa saadaan scene()-metodin avulla selville mihin GraphicsScene-olioon "tämä" hiiriolio kuuluu, ja sitten saadaan collidingItems()-metodin avulla lista niistä hiiriolioista joihin "tämä" hiiri törmää.

isinstance()-metodin avulla voidaan tutkia onko jokin olio jotain tiettyä tyyppiä. Tätä metodia on käytetty esim. Animals.py-ohjelmassa.

Tämän homman testaaminen saattaa olla helpompaa jos panee näkyville enemmän kuin yhden BigBlackMouse-olion.

Harjoitus 4:

Tutki GraphicsScene-luokan dokumentaatiota ja lisää ominaisuus että voit hiiren vasemmalla napilla lisätä hiiriä skeneen ja oikealla voit poistaa niitä skenestä. Poistamista varten ei tarvitse "osua" hiireen.