Python CLI -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
2022-12-28 Tiedosto luotu.

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.