Exercises related to Python programming

- When you write Python programs, it is important that you configure your program editor so that it does not put tabulator characters into the source program file. When you do these exercises, the settings of you program editor should be such that it puts three space charcters into the program file when the tabulator key is pressed.
- Because program blocks are formed through indentation in Python, tabulator characters can be harmful if tabulation step is not correct. The best way to get rid from all problems related to the tabulator character, is to not use that character in your souce program file. This way your program looks the same regardless of the editor or browser that is used to view the program.

Kari Laitinen
http://www.naturalprogramming.com
2009-10-05 File created.
2013-10-13 Last modification.

EXERCISES WITH PROGRAM GuessAWord.py

You can find a program named **GuessAWord.py** in the folder
`http://www.naturalprogramming.com/pythonprograms/pythonfilesextra/`
This program is a simple computer game in which the player has to try to guess the characters of a word that is ’known’ by the game. Study the program and play the game in order to find out how the game has been programmed.

Exercise 1:

Improve the Guess-A-Word game so that the word to be guessed is randomly taken from a list of words. A list of words can be created with a statement such as

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

A random index for a list such as the one above can be created with the `random.random()` method in an expression like

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

The `random.random()` method returns a `double` value in the range 0.0 .... 1.0 so that the value 1.0 is never returned. The above expression thus calculates a suitable random index. When `double` values are converted to `int` values, they are always rounded ’downwards’ to the smaller integer value. To use the `random.random()` method, you must have a suitable `import` statement in your program. (For advice, see the **MathDemo.py** program.)

Exercise 2:

Now the program is such that it terminates when the game is finished. Modify the program so that the game can be played several times during a single run of the program. In the above-mentioned folder there is a program named **RepeatableGame.py** which should be a helpful example.

Exercise 3:

Improve the program so that it counts how many guesses the player makes during a game. After a game is played, and before a new game starts, the program should print how many guesses were made. The following variable could be usedful in this task

```
    number_of_guesses = 0
```

Exercise 4:

Improve the program so that it prints game statistics before the program terminates. This means that the program shows which words were being guessed and how many guesses were made for each word. The game statistics could look like the following.

```
     PLAYED WORD      GUESSES

     COPENHAGEN         7
     LONDON             6
     COPENHAGEN         4
     BERLIN             5
     HELSINKI           4
```

As the ’played words’ will be randomly selected from an array, it is possible that the same word is played several times.

You can use the following kind of data items to store data of games:

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

A variable can be used to count how many games are played, and you can use lists to store the played words and the number of guesses made. A new data item can be added to the end of a list with a method named `append()`. New data should be put to the lists after each game is played, and the data should be displayed on the screen in the end when the user no longer wants to play new games.

EXERCISES WITH PROGRAM Animals.py

Exercise 1:

Write a new method named make_stomach_empty() to class Animal in Animals.py. The stomach of an Animal object can be emptied by writing an empty string "" as the stomach contents. The new method could be called

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

To test this new method, you should use method make_speak().

Exercise 2:

Modify the constructor of class Animal so that its parameter gets a default value. After this modification an Animal object can be created without supplying any parameters, for example, in the following way:

```
    default_animal  =  Animal()
```

The data field (instance attribute) species_name can be given the value "default animal".

By studying program Windows.py you can find out how constructor parameters can be given default values. Also in this exercise you should use method make_speak() to check that your modifications are correct.

Exercise 3:

The data members or data fields of Python objects are frequently called attributes or, more precisely, instance attributes. In Python classes, you do not need to declare the data fields. Instead, the data fields become existent when you write inside a method

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

Add a new data field (instance attribute) to class Animal by writing the statement

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

to the constructor. Here the aim is that an Animal object can be created, for example, with the statement

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

Copying of an object must be modified so that the new data field will be copied as well. Method make_speak() must be modified so that it produces an output that looks like

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

The new data field can be given the default value "nameless".

Exercise 4:

Modify method make_speak() so that it prints

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

in that case when stomach_contents references an empty string. The stomach of an Animal object is empty as long as method feed() has not been called. You can use standard function len() to check whether the stomach is empty. Function len() can be called, for example, in the following way.

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

       # stomach_contents references an empty string.
       ...
```

If the stomach is not empty, the program should produce the original output.

Exercise 5:

Modify method feed() so that it will be possible to feed another animal to an Animal object. Method feed() can operate in the same way as the constructor of the class: it can check the type of the received parameter, and then decide what to do. If a parameter of type Animal is given, the animal will be eaten.

Another animal can be "eaten", for example, so that the data field animal_name of the "eatable" animal will be copied to the stomach of the "eater". In the new feed() method, you can refer to the data field animal_name of the given parameter in the same way as is done in the constructor.

The data field animal_name of the "eaten" animal can be given the value "Eaten animal". When the new feed() method is written, the statements

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

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

should produce the output

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

Exercise 6:

Modify class Animal so that the data field stomach_contents stores a list of strings. In the constructor, the new kind of stomach can be created in the following way

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

The intention here is that when the feed() method is called, the given food (a string) is added to the end of this list. A new object can be added to a list with the append() method.

This modification requires that the constructor and methods of the class operate in a slightly different way. Method parameters, as well as the "main program" will remain unchanged.

When stomach contents is printed to the screen, the eaten strings should be read from the list. Strings stored in a list can be printed with a loop such as

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

    for string_to_print in list_of_strings :

       print string_to_print
```

EXERCISES WITH CLASS ISODate

The file ISODate.py contains a class named ISODate that can be used to make various calculations related to dates. This class is used in the programs Columbus.py, Birthdays.py, and Friday13.py.

The ISODate class handles date information in the so called ISO format, which means that dates are printed in format YYYY-MM-DD. (ISO is an abbreviation for International Organization for Standardization.)

Exercise 1:

Write a program that calculates your current age in years, months, and days. You can accomplish this when you first create ISODate objects in the following way

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

By studying program Columbus.py, you’ll find out how the time difference between two ISODate objects can be calculated. You can also easily find out on which day of the week you were born.

You can do this exercise so that you modify program Columbus.py. You should, however, change the names in the program so that they reflect what you are calculating. You can use the names given above.

Exercise 2:

Improve your program so that it prints a list of your most important birthdays and tells on which day of week those birthdays occur. You should study program Birthdays.py to find out how to do this.

Exercise 3:

In this exercise we'll study how a class can inherit another class in Python. Programs BankPolymorphic.py and Windows.py are examples in which inheritance is used.

Derive another class named AnotherDate from class ISODate. You can write the new AnotherDate class before the "main program" after the import statements. The AnotherDate class should be the same as class ISODate with the exception that it has an additional method that begins as follows

```
def to_anti_iso_format( self )  :
   # Here begin the internal statements of the method
```

This method should return a string that contains the date in anti-ISO format which is DD.MM.YYYY. You can create this method quite easily if you make of copy of method __str__(self), and change its name and internal statements. With this method you should be able to print dates in the following way

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

Note that when you derive a new class from an existing class in Python, the constructor method __init__(self) is also inherited. So you do not always need to write a constructor to a derived class.

Exercise 4:

Improve your program so that it prints dates when you are 10000 and 20000 days old. The age of a person is 10000 days, when his/her "conventional" age is approximately 27 years and 4 1/2 months. With this feature in your program, you’ll get new days for partying. This feature can be programmed when you increment a day counter and an ISODate object inside a loop, for example, in the following way.

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

while  ... :

   day_counter += 1
   date_to_increment.increment()

   if ...
```

Exercise 5:

Improve your program so that it tells when you are 1000000000 seconds old. Also this feature can be programmed so that you count days starting from your birthday. Each day has 24 * 60 * 60 seconds. 1000000000 seconds will be reached some time after you are 31 years old. Your program should print your age in years, months, and days on the day when you are 1000000000 seconds old. Again you’ll have one more day to celebrate!

First PyQt exercise: Drawing a Chessboard

With these exercises we’ll learn to use some drawing methods. You can do these exercises by modifying program HelloQt.py.

Exercise 1:

Study program DrawingDemoQt.py and find out how to draw a square to the drawing area.

Exercise 2:

When you know how to draw a square, modify your program so that it draws a chessboard that has 8 x 8 squares of which every second square is black and every second square is white. You do not have to draw the white squares as they can be of the background color. To draw the black squares, you should use loop(s).

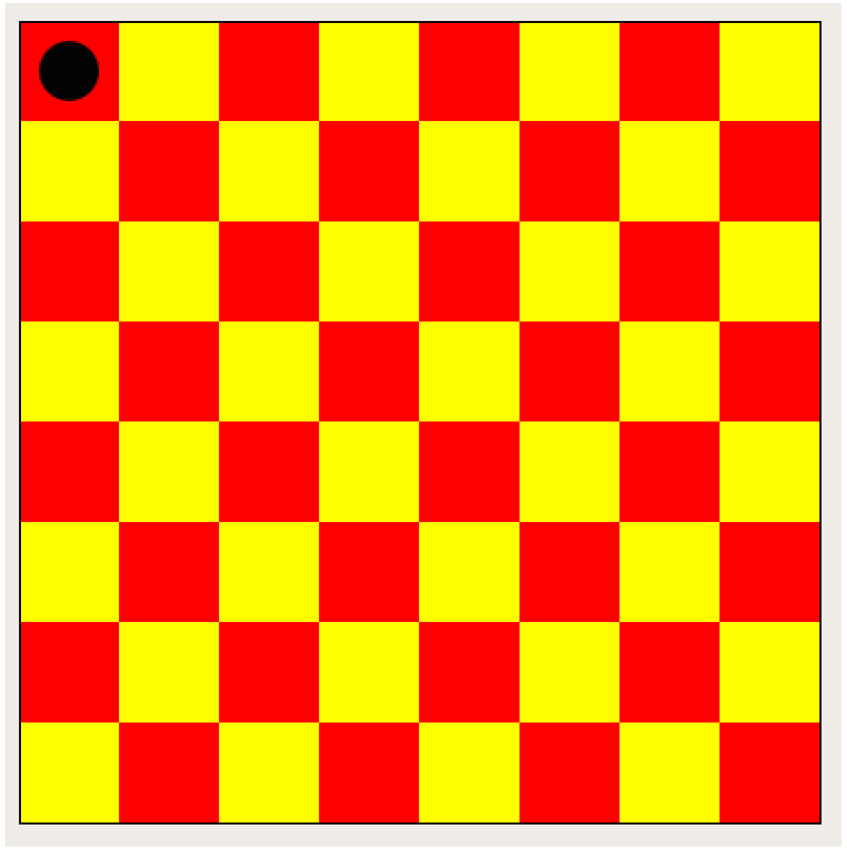
You should draw a frame around the chessboard at the end.

Exercise 3:

Find out how to set drawing colors in PyQt applications.
Draw the "black" squares of the chessboard with red color and "white" squares with yellow color. "White" squares can be drawn by first drawing a single large yellow square on which you later draw the "black" squares.

Exercise 4:

Draw a black ball (circle filled with black color) in the square which is in the upper left corner of the chessboard. The black ball represents a chess man on your chessboard. After this exercise is completed, your chessboard should resemble the chessboard below.



Exercise 5:

Among the basic Python programs, you can find a program named MathDemo.py which shows how to generate random numbers. Use random integers and put the "chess man" into a random square on the chessboard. Always when the program window is "repainted" the black ball should appear in a new random square.

Exercise 6:

Study the PyQt documentation and find out how the drawing color can be set by using numerical RGB values. When you know how to do this, modify your program so that the two chessboard colors, that have earlier been yellow and red, are replaced with random colors.

EXERCISES WITH PROGRAM AnimationDemoQt.py

Exercise 1:

Now the program is such that the blinking ball stays approximately at the center of the window. Modify the program so that the blinking ball starts moving gradually downwards. Always when the ball becomes visible after it had disappeared, it should be located a few pixels below its previous position. You should control that the y coordinate of the ball will not grow too large, so that the ball disappears entirely. After the ball has reached the bottom of the window, it should remain there and continue blinking.

You can modify the ball coordinate in the paintEvent() method after you have drawn the ball with the current coordinates.

Exercise 2:

Modify the program so that the ball starts moving gradually upwards after it has reached the bottom of the window. After the ball has reached the top of the window, it should start moving downwards again. The blinking ball should keep seesawing between the top and bottom of the window. You should add the following data field (instance attribute) to the class to control the ball movement:

```
    self.moving_down  =  True
```

This "variable" should be given value False when the ball starts moving upwards.

Exercise 3:

Modify the program so that it will move an image instead of a ball. By studying program SinglePictureQt.py, you’ll find out how an image can be drawn as a QImage object. Considering the following exercise, it is best to select an arrow image whose arrow points down. Arrow images can be found at http://www.oamk.fi/~karil/images/arrow_images/

Exercise 4:

If you were able to find an image that shows an arrow, you should modify the program so that the arrow points downwards when the image is moving downwards, and the arrow points upwards when the image is moving upwards. You can make this with a QImage method with which a mirrored version of an image can be created.

Exercise 5: (This may not be possible with QImage !!!!)

If you have an arrow image, you can modify the program so that the image travels around the borders of the window, the arrow always pointing to the direction of the movement.

EXERCISES WITH PROGRAM MouseDemoQt.py

MouseDemoQt.py is a program that demonstrates how to react to the pressings of mouse buttons, mouse movements, etc. in Python Qt programs. Study that program first, and modify it as required in the following exercises.

Exercise 1:

Modify the program so that it will draw an image to the screen when it starts executing. You can remove the program lines that perform the original drawing operations in the program. By studying program SinglePictureQt.py you’ll find out how to draw images.

Exercise 2:

After your program is able to show a picture, improve it so that it is possible to "drag" the picture with the mouse. The intention is that it should be possible to move the picture with the left mouse button according to the following rules

- when the left mouse button is pressed down while the cursor is over the picture, the movement operation begins, and the cursor coordinates are stored into data fields (instance attributes) inside the class
- when the mouse is moved while the left mouse button is pressed down, the program calculates how much the mouse moved, and moves the picture accordingly
- when the left mouse button is released, the movement operation is complete, and the picture stays in its final position

The following data field (instance attribute) might be useful in your program.

```
    self.picture_is_being_moved  =  False
```

This could get the value True while the picture is being moved. With this kind of data field it is easy to control the situation when the mouse button is pressed down outside the picture area and released inside the picture area.

The following source program lines could be used to check whether a clicked point is inside the area of a picture:

```
   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.
```

Exercise 3:

Modify the program further so that a slightly darkened version of the picture will be displayed while it is being moved. This small modification will most likely improve the user experience. This can be achieved by drawing a rectangle over the picture with an almost-transparent dark color.

Exercise 4:

Modify the program so that it will be possible to use the right mouse button on alter the size of the shown picture. When the right mouse button is pressed down while the cursor is inside the picture area, the size of the picture should grow a little bit. If the Ctrl key is pressed down while the right mouse button is used, the size of the picture should be decreased.

The QImage class provides a method to create an enlargened or shrinked version of an image. However, it may be better to control the picture size programmatically so that the class has data members which store the desired picture size. The drawImage() method of the QPainter class can then be used to draw the image by using the desired size.

EXERCISES WITH PROGRAM FlagsQt.py

FlagsQt.py is an example program that has been constructed in an object-oriented way, using several classes. It demonstrates the use of the QTabWidget class with which it is possible to construct a UI that consists of several tabbed pages or tabbed widgets.

Exercise 1:

Add a new flag to one of the tabbed pages, which are FlagPanel objects in FlagsQt.py. The new flag can be a flag of a fictitious country. For example, the flag of Absurdistan might have all rainbow colors as vertical stripes. Such a flag can be easily constructed by using the VerticalStripesFlag class.

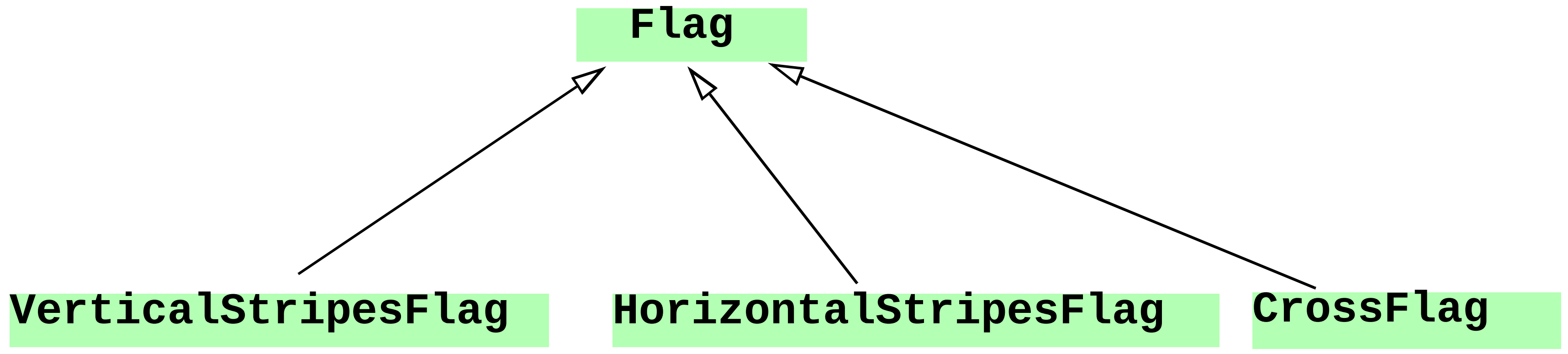
Exercise 2:

Add a new tabbed page "Fictitious" to the UI. This new page can be created in the same way as the other pages are created, using the class FlagPanel. You should add the new flag that you created in the previous exercise to the new page.

It is possible that the program will not work if no flags are added to the new FlagPanel.

Exercise 3:

In the FlagsQt.py program, class Flag serves as a superclass for several subclasses whose objects represent flags in a certain category. The hierarchy of classes is the following.



Each subclass has its own draw() method that is able to draw the flag represented by the class. A correct draw() method is automatically selected for each flag object that is stored in a list of flag objects.

In this exercise you should derive a new class from class Flag. The name of the new class might be BallFlag as its objects should represent flags that have a ball shape in the middle of the flag. For example, the flag of Japan has a red ball on white background, and the flag of Bangladesh has a reddish ball on green background. The draw() method of the new class should be able to draw a ball in the middle of the flag background. (You can use the flag of Bangladesh as an example here, although officially the ball in the Bangladesh flag is not exactly in the middle of the flag.) To test your new class, you should add BallFlag objects to your program.

Exercise 4:

In this exercise you should derive a yet new class from the Flag class. The name of the new class might be SingleStarFlag as the new class should represent flags that have a single five-pointed star in the middle of the flag. The flags of Vietnam, Somalia, and Morocco are like this, although the star in these flags is not necessarily of the same size.

To draw the star shape you should use the QPainterPath class which is used, for example, in program StarsQt.py. Actually, the StarsQt.py program already has a class named StarShape5 that represents a five-pointed star. The star of your SingleStarFlag class can be defined as an object, and it can be drawn by using the draw() method in the StarShape5 class. You can use the StarShape5 class in StarsQt.py, if you first copy the file StarsQt.py to the same folder where your program is, and then use the following import statement in your program:

```
    from StarsQt import StarShape5
```

Again, you need to create SingleStarFlag objects in order to test the new feature of your program.

Exercise 5:

A nice improvement in this program might be a FlagDesignPanel that could be added to the QTabWidget in the same way as FlagPanel objects are added. With the FlagDesignPanel one could design new flags. Consult the teacher if you would like to make such a panel to your program.