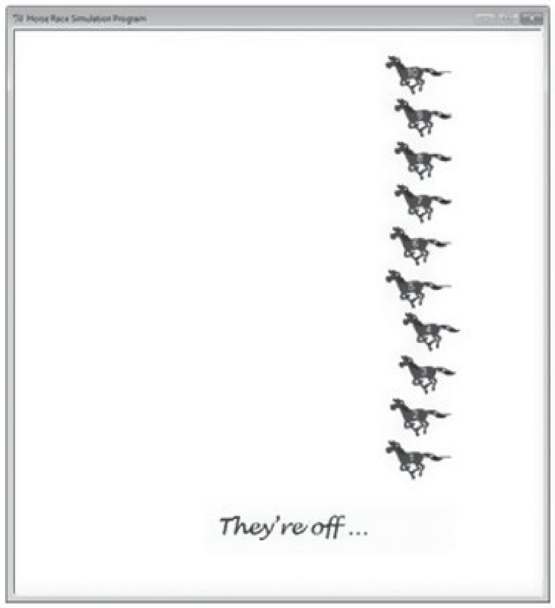
Horse Race Simulation Program

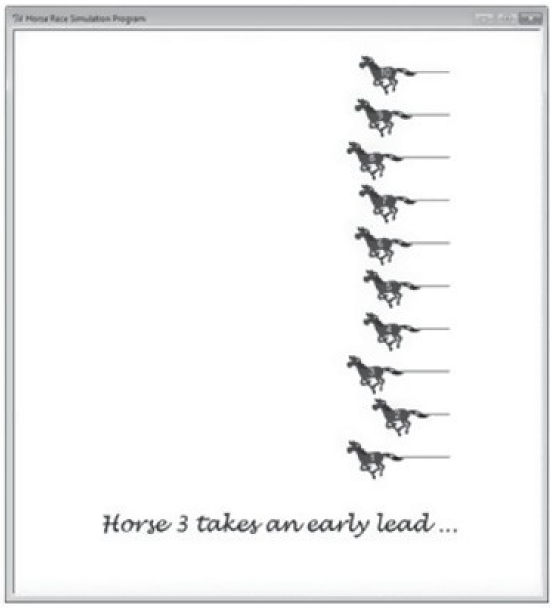
In this program, you will design, implement and test a program that simulates a horse race.

The Problem

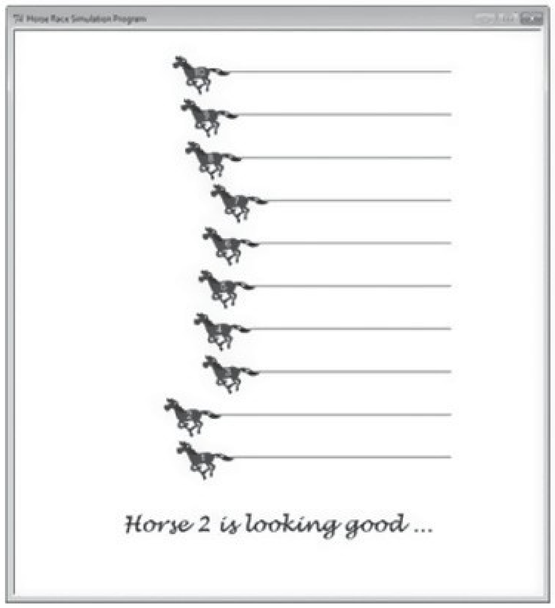
The problem is to create a visualization of a horse race in which horses are moved ahead a random distance at fixed intervals until there is a winner, as seen below:



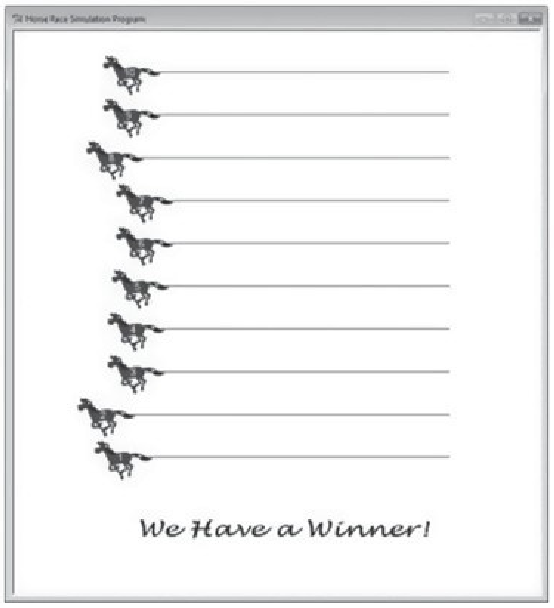
They're off ...



Horse 3 takes an early lead ...



Horse 2 is looking good ...



We Have a Winner!

Problem Analysis

The program needs a source of random numbers for advancing the horses a random distance in the race. We can use the random number generator of the Python standard library module `random` that you used Coin Change Exercise in Unit 3. The remaining part of the problem is in the creation of appropriate graphics for producing a visualization of a horse race. You will make use of the turtle graphics module from the Python standard library to do this.

Program Design

Meeting the Program Requirements

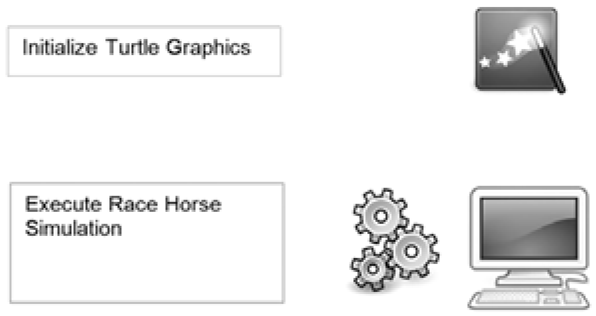
There are no specific requirements for this problem, other than to create an appropriate simulation of a horse race. Therefore, the requirement is essentially the generation of a horse race in which the graphics look sufficiently compelling, and each horse has an equal chance of winning a given race. Since a specific number of horses was not specified, we will design the program for ten horses in each race.

Data Description

The essential information for this program is the current location of each of the ten horses in a given race. Each turtle is an object, whose attributes include its shape and its coordinate position on the turtle screen. Therefore, we will maintain a list of ten turtle objects with the shape attribute of a horse image for this purpose. Thus, suitable horse images must be found or created for this purpose.

Overall Program Steps

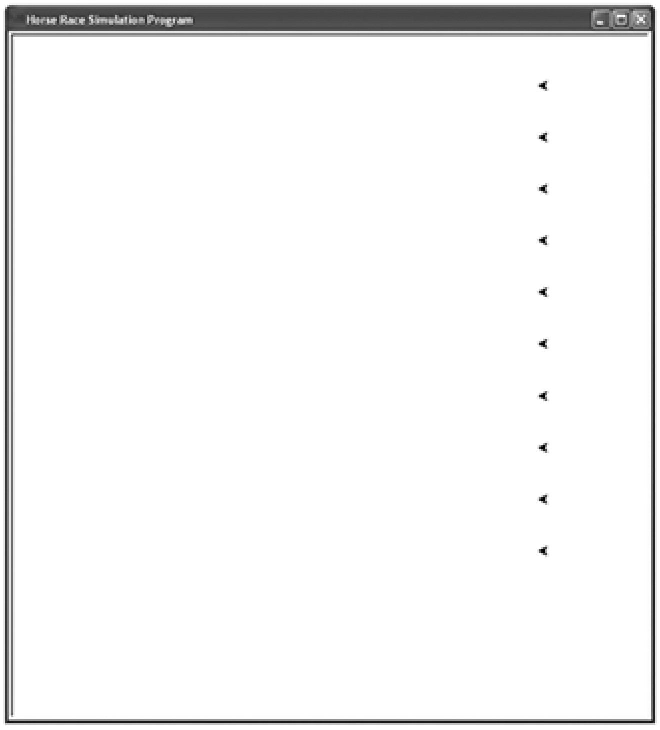
The overall steps in this program design are given in Figure 6-31.



Program Implementation and Testing

Stage 1 - Creating an Initial Turtle Screen Layout

You will first develop and test an initial program that lays out the positions of the starting horses on the turtle graphics screen, as shown in Figure 6-32. Figure 6-33 provides this first stage of the program.



Enter the Stage 1 Code from the sample on the next page.

```
 1  # Horse Racing Program (Stage 1)
 2
 3  import turtle
 4
 5  def newHorse():
 6      horse = turtle.Turtle()
 7      return horse
 8
 9  def generateHorses(num_horses):
10      horses = []
11      for k in range(0, num_horses):
12          horse = newHorse()
13          horses.append(horse)
14
15      return horses
16
17  def placeHorses(horses, loc, separation):
18      for k in range(0, len(horses)):
19          horses[k].hideturtle()
20          horses[k].penup()
21          horses[k].setposition(loc[0], loc[1] + k * separation)
22          horses[k].setheading(180)
23          horses[k].showturtle()
24
25  # ---- main
26
27  # init number of horses
28  num_horses = 10
29
30  # set window size
31  turtle.setup(750, 800)
32
33  # get turtle window
34  window = turtle.Screen()
35
36  # set window title bar
37  window.title('Horse Race Simulation Program')
38
39  # init screen layout parameters
40  start_loc = (240, -200)
41  track_separation = 60
42
43  # generate and init horses
44  horses = generateHorses(num_horses)
45
46  # place horses at starting line
47  placeHorses(horses, start_loc, track_separation)
48
49  # terminate program when close window
50  turtle.exitonclick()
```

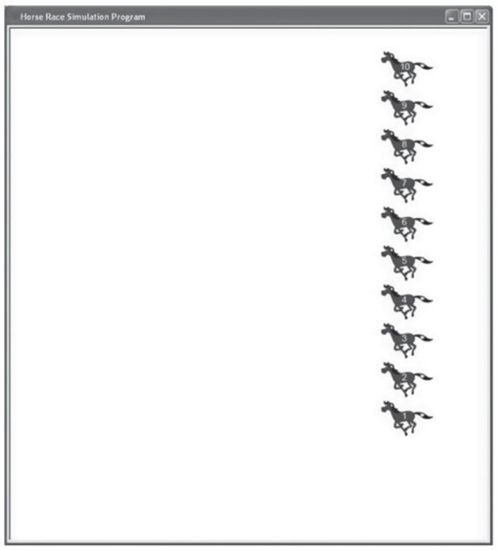
Notes:
At line 3 the turtle module is imported. Since the `import` *`module_name`* form of import is used, each call to a method of this module must be prefixed with the module name. For example, `turtle. setup(750, 800)` on line 31 (which sets the turtle screen size to a width of 750 and a height of 800 pixels).

The intent of this version of the program is to ensure that the turtle screen is appropriately sized and that the initial layout of horse locations is achieved. Therefore, only the default turtle shape is used at this point. In the next version we will focus on generating a set of horse images on the screen. Thus, on line 34, the turtle screen object is retrieved (by the call to `turtle. Sreen()`) and its reference assigned to variable `window.` The start location of the first (lowest) horse is set to an x coordinate value of 240, and a y coordinate value of 2200. This puts the turtle screen object at the lower right corner of the screen. The amount of vertical separation between the horses is assigned to variable `track_separation.` These values were determined from knowledge of the screen coordinates in turtle graphics and a little trial and error.

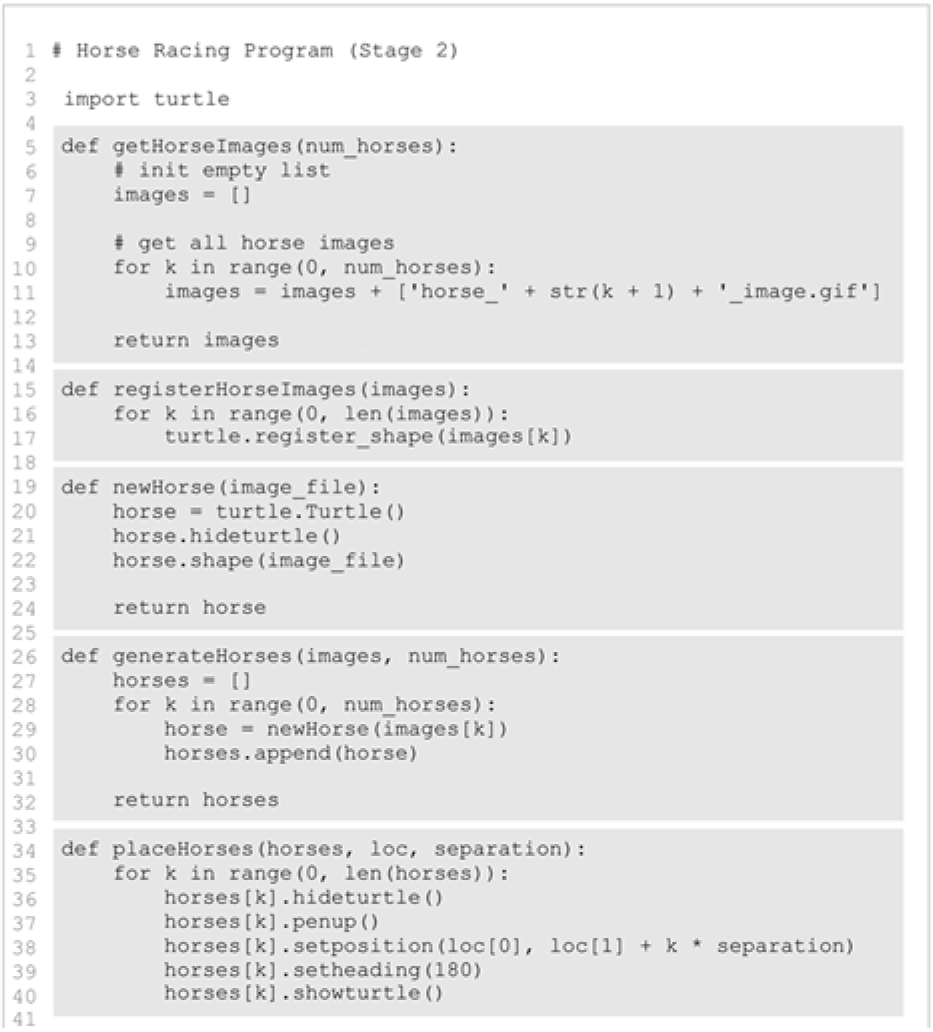
Next, on line 44 a call is made to function `generateHorses` (at lines 9–15). This function returns a list of ten new turtle objects, and assigned to variable `horses`. Function `newHorse` (lines 5–7) is called by function `generateHorses` to create each new horse turtle object. At this stage, function `newHorse` simply creates and returns a regular turtle object. In the next stage how- ever, it will be responsible for returning new turtle objects with an appropriate horse shape.
The position for each of these horses is determined by function `placeHorses` on lines 17–23. It is passed the list of horse turtle objects, the location of the first turtle, and the amount of separation between each (established as 60 pixels on line 41). Function `placeHorses`, therefore, contains a for loop that iterates over the list of horse objects and makes them initially hidden with their pen up (lines 19–20), moves each to its starting position (line 21), sets the heading of each to 180 degrees to move left (line 22), and then makes each visible (line 23). Finally, method `exiton-click()` is called so that the program will terminate when the user clicks on the program window's close box.

Stage 2 - Adding the Appropriate Shapes and Images

We next develop and test the program with additional code that adds the horse shapes (images) needed. The resulting turtle screen is shown below.



Enter the Stage 2 Code from the sample below:



```
 1 # Horse Racing Program (Stage 2)
 2
 3  import turtle
 4
 5 def getHorseImages(num_horses):
 6     # init empty list
 7     images = []
 8
 9     # get all horse images
10     for k in range(0, num_horses):
11         images = images + ['horse_' + str(k + 1) + '_image.gif']
12
13     return images
14
15 def registerHorseImages(images):
16     for k in range(0, len(images)):
17         turtle.register_shape(images[k])
18
19 def newHorse(image_file):
20     horse = turtle.Turtle()
21     horse.hideturtle()
22     horse.shape(image_file)
23
24     return horse
25
26 def generateHorses(images, num_horses):
27     horses = []
28     for k in range(0, num_horses):
29         horse = newHorse(images[k])
30         horses.append(horse)
31
32     return horses
33
34 def placeHorses(horses, loc, separation):
35     for k in range(0, len(horses)):
36         horses[k].hideturtle()
37         horses[k].penup()
38         horses[k].setposition(loc[0], loc[1] + k * separation)
39         horses[k].setheading(180)
40         horses[k].showturtle()
41
```

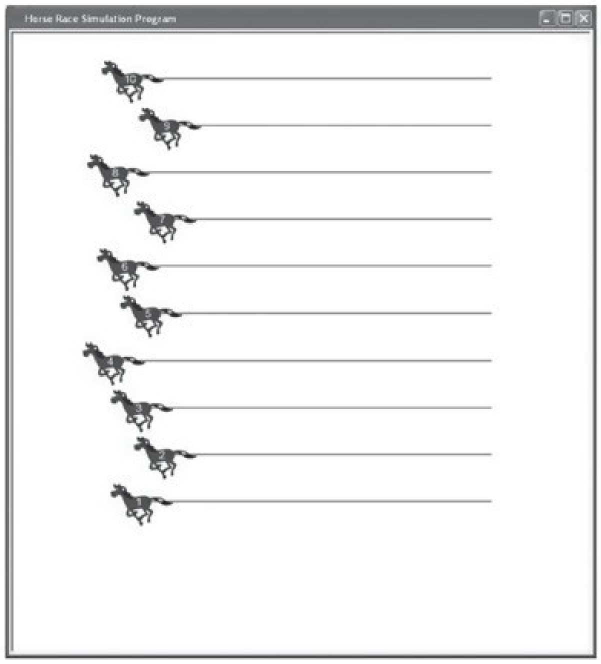
Notes:

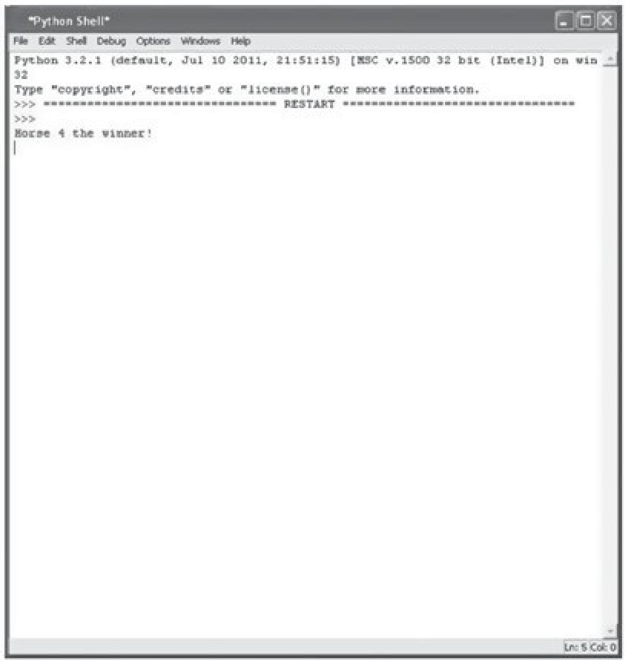
In this stage of the program we add functions `getHorseImages` and `registerHorseImages`, called from lines 61 and 62 of the main program section. Function `getHorseImages` returns a list of GIF image files. Each image contains the same horse image, each with a unique number 1 to 10. Function `registerHorseImages` does the required registering of images in turtle graphics by calling method `turtle.register_shape` on each.

Function `generateHorses` (lines 26–32) is implemented the same way as in stage 1 to return a list of horse turtle objects, except that it is altered to be passed an argument containing a list of horse images. Thus, the call to `generateHorses` in line 65 is altered to pass the list of images in variable `horse_images`. Function new`Horse` (lines 19–24) is altered as well to be passed a particular horse image for the horse that is created, `horse.shape(image_file)`.

Stage 3—Animating the Horses

Next you will develop and test the program with additional code that animates the horses so that they are randomly advanced until a horse crosses the finish line. The resulting turtle screen is shown below:





```
Python 3.2.1 (default, Jul 10 2011, 21:51:15) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Horse 4 the winner!
```

Enter the Stage 3 Code from the sample on the **next two pages**.

Refer to the notes that follow to guide you.

```python
# Horse Racing Program (Stage 3)

import turtle
import random

def getHorseImages(num_horses):
    # init empty list
    images = []

    # get all horse images
    for k in range(0, num_horses):
        images = images + ['horse_' + str(k+1) + '_image.gif']

    return images

def registerHorseImages(images):
    for k in range(0, len(images)):
        turtle.register_shape(images[k])

def newHorse(image_file):
    horse = turtle.Turtle()
    horse.hideturtle()
    horse.shape(image_file)

    return horse

def generateHorses(images, num_horses):
    horses = []
    for k in range(0, num_horses):
        horse = newHorse(images[k])
        horses.append(horse)

    return horses

def placeHorses(horses, loc, separation):
    for k in range(0, len(horses)):
        horses[k].hideturtle()
        horses[k].penup()
        horses[k].setposition(loc[0], loc[1] + k * separation)
        horses[k].setheading(180)
        horses[k].showturtle()
        horses[h].pendown()

def startHorses(horses, finish_line, forward_incr):
    # init
    have_winner = False

    k = 0
    while not have_winner:
        horse = horses[k]
        horse.forward(random.randint(1, 3) * forward_incr)

        # check for horse over finish line
        if horse.position()[0] < finish_line:
            have_winner = True
        else:
            k = (k + 1) % len(horses)
    return k

```

```
60  def displayWinner(winning_horse):
61      print('Horse', winning_horse, 'the winner!')
62
63  # ---- main
64
65  # init number of horses
66  num_horses = 10
67
68  # set window size
69  turtle.setup(750, 800)
70
71  # get turtle window
72  window = turtle.Screen()
73
74  # set window title bar
75  window.title('Horse Race Simulation Program')
76
77  # init screen layout parameters
78  start_loc = (240, -200)
79  finish_line = -240
80  track_separation = 60
81  forward_incr = 6
82
83  # register images
84  horse_images = getHorseImages(num_horses)
85  registerHorseImages(horse_images)
86
87  # generate and init horses
88  horses = generateHorses(horse_images, num_horses)
89
90  # place horses at starting line
91  placeHorses(horses, start_loc, track_separation)
92
93  # start horses
94  winner = startHorses(horses, finish_line, forward_incr)
95
96  # display winning horse
97  displayWinner(winner + 1)
98
99  # terminate program when close window
100 turtle.exitonclick()
```

Notes:

Two new functions are added in this version of the program, `startHorses` and `display-Winner`. Function `startHorses` (lines 44–58) is passed the list of horse turtle objects, the location of the finish line (as an x coordinate value on the turtle screen) and the fundamental increment amount - each horse is advanced by one to three times this amount. The while loop for incrementally moving the horses is on line 49. The loop iterates until a winner is found, that is, until the variable `have_winner` is `True.` Therefore, `have_winner` is initialized to `False` in line 46. Variable `k`, initialized on line 48, is used to index into the list of horse turtle objects.

Since each horse in turn is advanced some amount during the race, variable `k` is incremented by one, modulo the number of horses in variable `num_horses` (10) (line 57). When `k` becomes equal to `num_horses 21` (9), it is reset to 0 (for horse 1).

The amount that each horse is advanced is a factor of one to three randomly determined by call to method `randint(1,3)` of the Python standard library module `random` in line 51. Variable `forward_incr` is multiplied by this factor to move the horses forward an appropriate amount.

The value of `forward_incr` is initialized in the main program section. This value can be adjusted to speed up or slow down the overall speed of the horses. Function `displayWinner` displays the winning horse number in the Python shell (lines 60–61). This function will be rewritten in the next stage of program development to display a “winner” banner image in the turtle screen. Thus, this implementation of the function is for testing purposes only.

The main program section (lines 63–100) is the same as in the previous stage of program development, except for the inclusion of the calls to functions `startHorses` and `display-Winner` in lines 94 and 97.

Final Stage—Adding Race Banners

Finally, we add the code for the displaying of banners at various points in the race as shown on page 1.

Enter the Final Code from the sample on the **next three pages**.

Refer to the notes that follow to guide you.

```python
# Horse Racing Program (Final Stage)

import turtle
import random
import time

def getHorseImages(num_horses):
    # init empty list
    images = []

    # get all horse images
    for k in range(0, num_horses):
        images = images + ['horse_' + str(k + 1) + '_image.gif']

    return images

def getBannerImages(num_horses):
    # init empty list
    all_images = []

    # get "They're Off" banner image
    images = ['theyre_off_banner.gif']
    all_images.append(images)

    # get early lead banner images
    images = []
    for k in range(0, num_horses):
        images = images + ['lead_at_start_' + str(k + 1) + '.gif']
    all_images.append(images)

    # get mid-way lead banner images
    images = []
    for k in range(0, num_horses):
        images = images + ['looking_good_' + str(k + 1) + '.gif']
    all_images.append(images)

    # get "We Have a Winner" banner image
    images = ['winner_banner.gif']
    all_images.append(images)

    return all_images

def registerHorseImages(images):
    for k in range(0, len(images)):
        turtle.register_shape(images[k])

```

```
def registerBannerImages(images):
    for k in range(0, len(images)):
        for j in range(0, len(images[k])):
            turtle.register_shape(images[k][j])

def newHorse(image_file):
    horse = turtle.Turtle()
    horse.hideturtle()
    horse.shape(image_file)

    return horse

def generateHorses(images, num_horses):
    horses = []
    for k in range(0, num_horses):
        horse = newHorse(images[k])
        horses.append(horse)

    return horses

def placeHorses(horses, loc, separation):
    for k in range(0, len(horses)):
        horses[k].hideturtle()
        horses[k].penup()
        horses[k].setposition(loc[0], loc[1] + k * separation)
        horses[k].setheading(180)
        horses[k].showturtle()
        horses[k].pendown()

def findLeadHorse(horses):
    # init
    lead_horse = 0

    for k in range(1, len(horses)):
        if horses[k].position()[0] <  \
           horses[lead_horse].position()[0]:
            lead_horse = k
    return lead_horse

def displayBanner(banner, position):
    the_turtle = turtle.getturtle()
    the_turtle.setposition(position[0], position[1])
    the_turtle.shape(banner)
    the_turtle.stamp()

```

```python
def startHorses(horses, banners, finish_line, forward_incr):
    # init
    have_winner = False
    early_leading_horse_displayed = False
    midrace_leading_horse_displayed = False

    # display "They’re Off" banner image
    displayBanner(banner_images[0][0], (70, -300))

    k = 0
    while not have_winner:
        horse = horses[k]
        horse.forward(random.randint(1, 3) * forward_incr)

        # display mid-race lead banner
        lead_horse = findLeadHorse(horses)
        if horses[lead_horse].position()[0] < -125 and \
           not midrace_leading_horse_displayed:

            displayBanner(banners[2][lead_horse], (40, -300))
            midrace_leading_horse_displayed = True

        # display early lead banner
        elif horses[lead_horse].position()[0] < 125 and \
           not early_leading_horse_displayed:
            displayBanner(banners[1][lead_horse], (10, -300))
            early_leading_horse_displayed = True

        # check for horse over finish line
        if horse.position()[0] < finish_line:
            have_winner = True
        else:
            k = (k + 1) % len(horses)
    return k
```

```
def displayWinner(winning_horse, winner_banner):
    # display "We Have a Winner" banner
    displayBanner(winner_banner, (20, -300))

    # blink winning horse
    show = False
    blink_counter = 5
    while blink_counter != 0:
        if show:
            winning_horse.showturtle()
            show = False
            blink_counter = blink_counter - 1
        else:
            winning_horse.hideturtle()
            show = True

        time.sleep(.4)

# ---- main

# init number of horses
num_horses = 10

# set window size
turtle.setup(750, 800)

# get turtle window
window = turtle.Screen()

# set window title
window.title('Horse Race Simulation Program')

# hide default turtle and keep from drawing
the_turtle.hideturtle()
the_turtle.penup()

# init screen layout parameters
start_loc = (240, -200)
finish_line = -240
track_separation = 60
forward_incr = 6

# register images
horse_images = getHorseImages()
banner_images = getBannerImages()
registerHorseImages(horse_images)
registerBannerImages(banner_images)

# generate and init horses
horses = generateHorses(horse_images)

# place horses at starting line
placeHorses(horses, start_loc, track_separation)

# start horses
winner = startHorses(horses, banner_images, finish_line,
                     forward_incr)

# light up for winning horse
displayWinner(horses[winner], banner_images[3][0])

# terminate program when close window
turtle.exitonclick()
```

Notes:

This final version imports one additional module, Python Standard Library module `time` (line 5), used to control the blink rate of the winning horse.

While the race progresses within the while loop at line 102, checks for the location of the lead horse are made in two places - before and after the halfway mark of the race (on line 108). If the *x* coordinate location of the lead horse is less then 125, the “early lead banner” is displayed on line 117 by a call to function `displayBanner`. Otherwise, if one second has elapsed, then the “midrace lead banner” is displayed on line 111.

The `sleep` method of the time module is used to control the blinking of the winning horse in function `displayWinner`. A “count-down” variable, `blink_counter`, is set to 5 on line 133. This will cause the winning horse to blink five times. The following while loop decrements `blink_counter` and continues to iterate until `blink_counter` is 0. Variable `show`, initialized to `False` on line 132, is used to alternately show and hide the turtle based on its current (Boolean) value, which is toggled back and forth between `True` and `False` each time through the loop. The sleep method is called on line 143 to cause the program execution to suspend for four-tenths of a second so that the switch between the visible and invisible horse appears slowly enough to cause a blinking effect. This version of `displayWinner` replaces the previous version that simply displayed the winning horse number in the Python shell window.

Added functions `getBannerImages` (lines 17–41), `registerBannerImages` (lines 47–50), and `displayBanner` (lines 86–90) incorporate the banner images into the program the same way that the horse images were incorporated in the previous program version. Function `startHorses` was modified to take another parameter, `banners`, containing the list of registered banners displayed during the race, passed to it from the main program section.

Finally, the default turtle (created with the turtle graphics window) is utilized in function `displayBanners` and in the main section. It is used to display the various banners at the bottom of the screen. To do this, the turtle’s “shape” is changed to the appropriate banner images stored in list `banner_images`. To prevent the turtle from drawing lines when moving from the initial `(0, 0)` coordinate location to where banners are displayed, the default turtle is hidden and its pen attribute is set to “up” (lines 160–161).