Horse Race Simulation Handout

- In Lab 9 Problem 6 we're going to **extend** a program that simulates a horse-race.
- You are provided with the base-line source-code on LearnUs.
  - Please download file HorseRacing.zip and set it up as a project in PyCharm.
- Please read the following slides to understand the provided program.
- Please refer to Lab 9 Problem 6 on how to extend this program.

Horse Race Simulation

The Problem

The problem is to create a visualization of a horse race in which horses are moved ahead a random distance at fixed time intervals until there is a winner.

Horse Race Simulation

Problem Analysis

The program needs a source of random numbers for advancing the horses a random distance in the race. We can use a random number generator of the Python standard library module `random`.

There must also be a way to control the pace of the race, so that the horses don't move across the screen too quickly. For this we can make use of Python standard library module `time`.

The remaining part of the problem is the creation of appropriate graphics for producing a visualization of a horse race. We shall make use of the turtle graphics module from the Python standard library.

Horse Race Simulation

Program Design

- **Meeting the Program Requirements**
- **Data Desciption**
- **Algorithmic Approach**

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 horse races 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.

Algorithmic Approach

There is no algorithm, per se, needed in this program other than to advance each horse a random distance at fixed time intervals until one of the horses reaches a certain point on the turtle screen (the “finish line”).

Initialize Turtle Graphics

Execute Race Horse Simulation

The Overall Steps of the Program

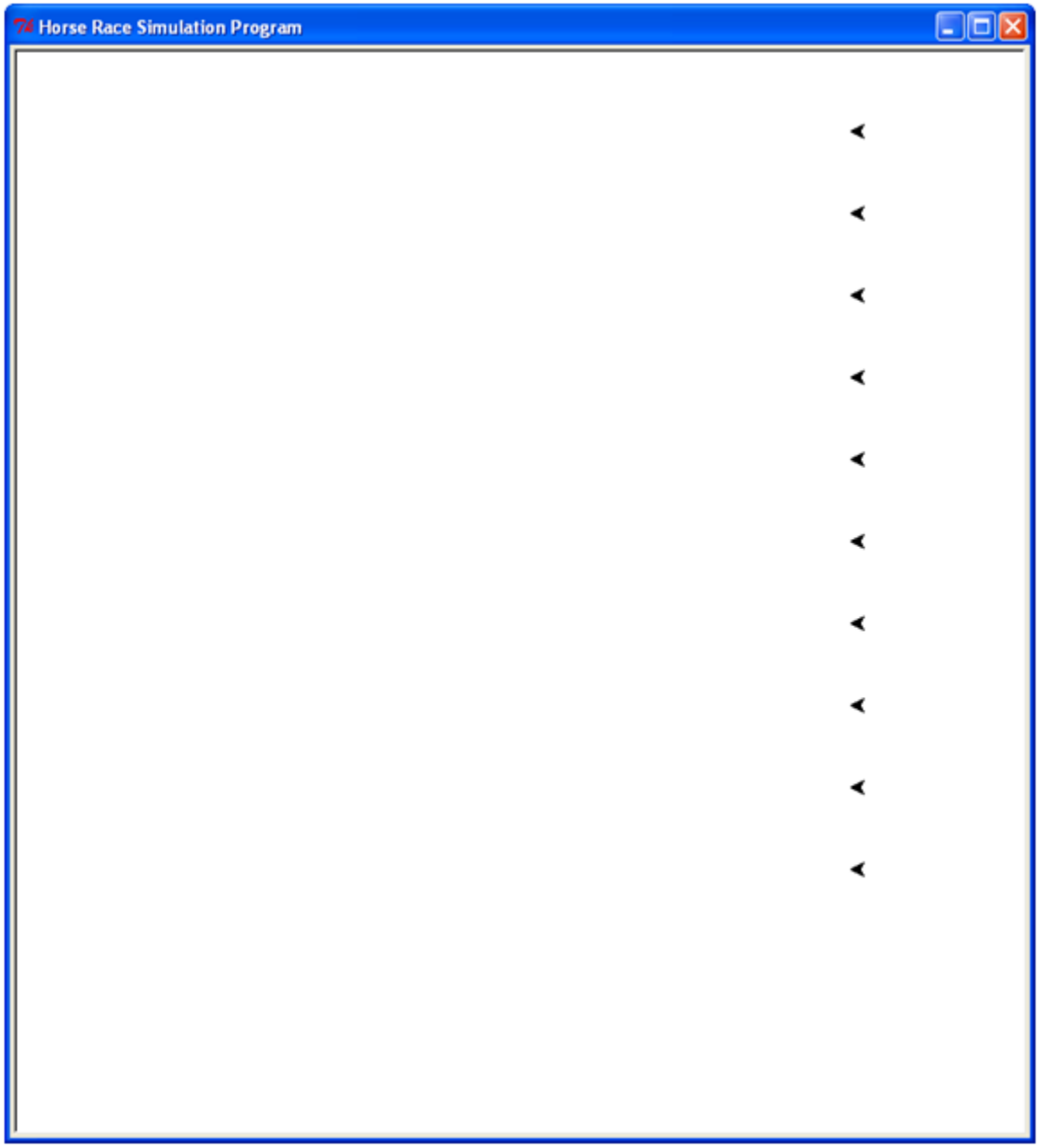
Horse Race Simulation

Program Implementation

Stage 1— Creating an Initial Turtle Screen Layout

We first develop and then test an initial program that lays out the starting positions of the horses on the turtle graphics screen.

The extent 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, this version simply uses the default turtle image. In the next version we will focus on generating a set of horse images on the turtle screen.


Horse Race Simulation Program

```
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()
```

On **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.

On **line 40**, the coordinates of the first (lowest) horse displayed are set. The vertical separation of horses is assigned to `track_separation`.

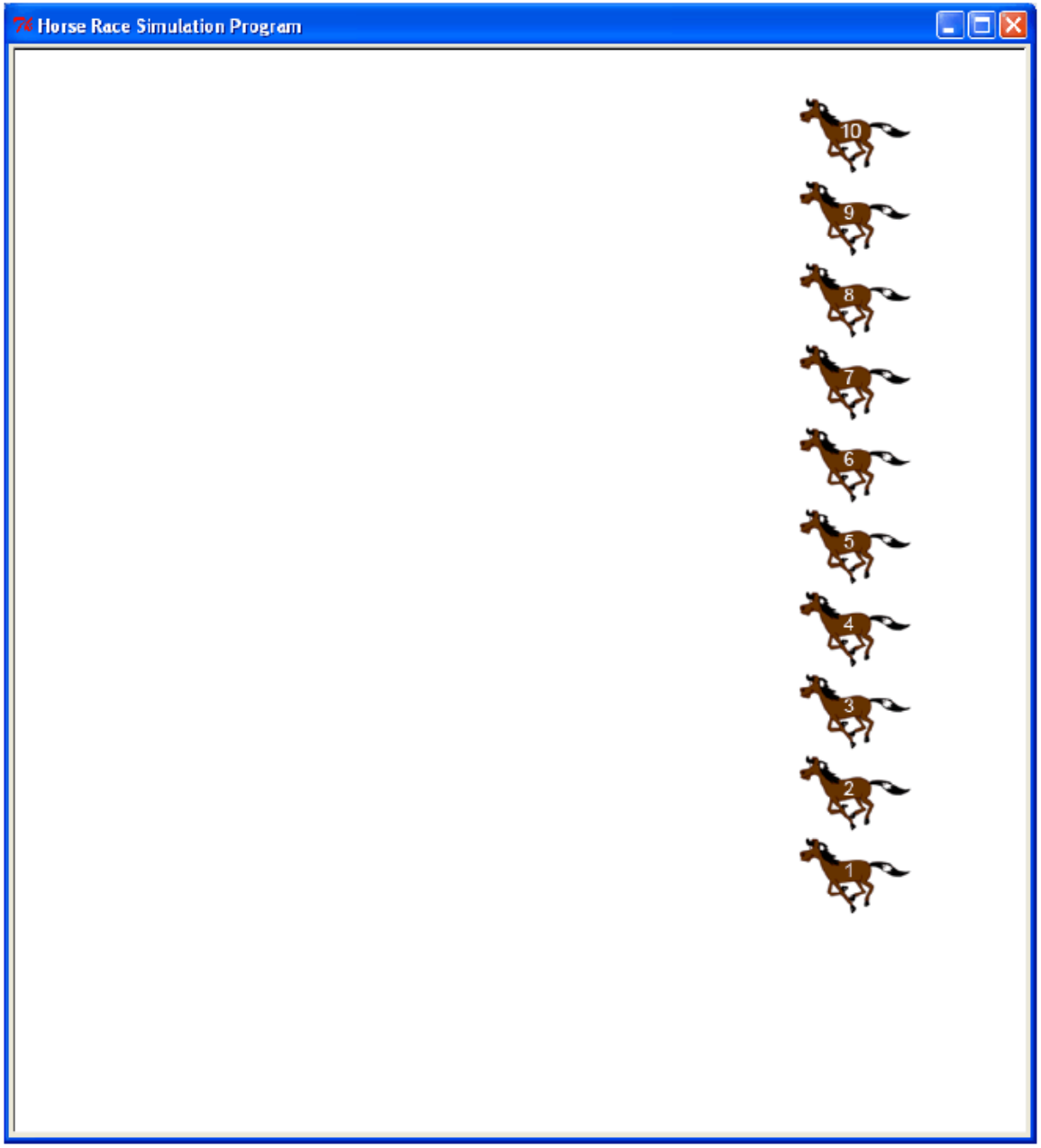
Function `generateHorses`, called on **line 44**, returns a list of ten new turtle objects, assigned to variable `horses`. Function `newHorse` (**lines 5–7**) is called to create each new turtle object (at this stage returning a regular turtle shape).

Function `placeHorses` (**lines 17–23**) is passed the list of turtle objects, the location of the first turtle, and the amount of separation between each and determines the position of each (established as 60 pixels on **line 41**). Each horse is initially hidden with pen up (**lines 19–20**), placed at its starting position (**line 21**), heading left (**line 22**), and made visible (**line 23**). Finally, method `exitonclick()` (**line 50**) is called so that the program will terminate when the user clicks on the program window's close box.

Program Implementation

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.


Horse Race Simulation Program
10
9
8
7
6
5
4
3
2
1

```
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
```

On **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. (The use of Python modules is covered in Chapter 7).

We add in this stage of the program functions `getHorseImages` (**lines 5-15**) and `registerHorseImages` (**lines 15-17**) (called from **lines 61- 62** of main). Function `getHorseImages` returns a list of GIF image files, each image the same horse image, with a unique number from 1 to 10 added. Function `registerHorseImages` does the required registering of images.

Function `generateHorses` (**lines 26–32**) is the same as in stage 1, except that it is passed a list of horse images rather than the number of horses to generate..

Function `newHorse` (**lines 19–24**) is altered as well to be passed a particular horse image to set the shape of the turtle object that this horse created, `horse.shape(image_file)`.

```
42 # ---- main
43
44 # init number of horses
45 num_horses = 10
46
47 # set window size
48 turtle.setup(750, 800)
49
50 # get turtle window
51 window = turtle.Screen()
52
53 # set window title bar
54 window.title('Horse Race Simulation Program')
55
56 # init screen layout parameters
57 start_loc = (240, -200)
58 track_separation = 60
59
60 # register images
61 horse_images = getHorseImages()
62 registerHorseImages(horse_images)
63
64 # generate and init horses
65 horses = generateHorses(horse_images)
66
67 # place horses at starting line
68 placeHorses(horses, start_loc, track_separation)
69
70 # terminate program when close window
71 turtle.exitonclick()
```

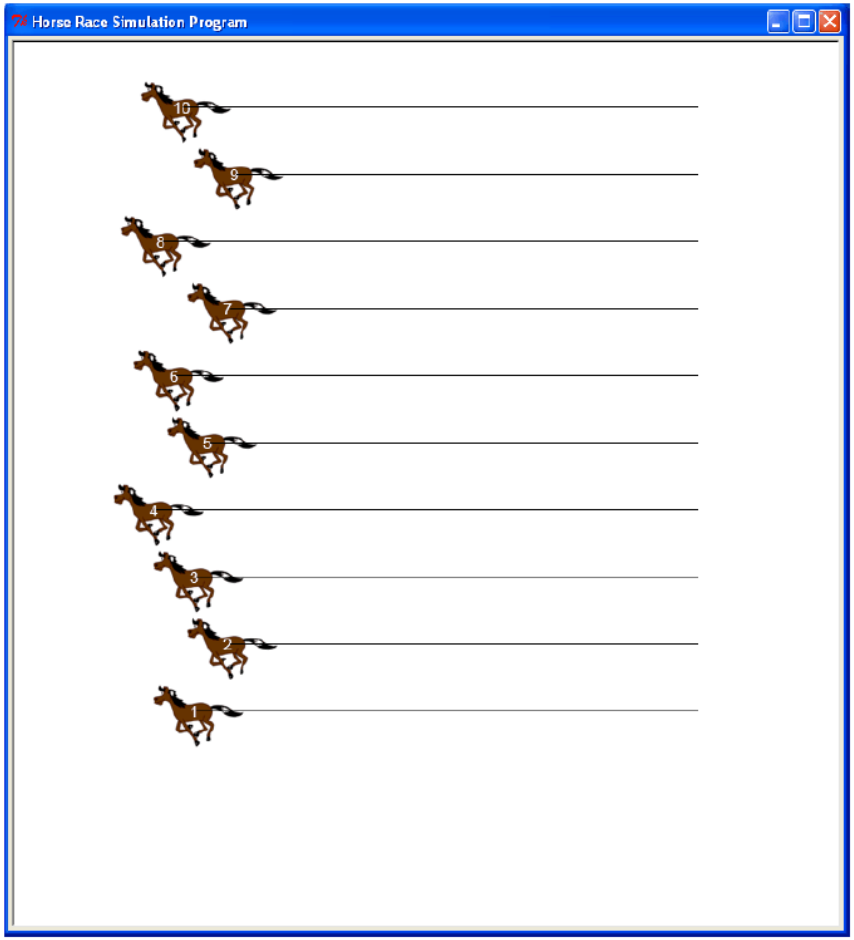
In this stage of the program we add functions `getHorseImages` and `registerHorseImages`, called from **lines 61** and **62**. Function `getHorseImages` returns a list of GIF image files. Each image contains the same horse image, each with a unique number from 1 to 10. Function `registerHorseImages`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

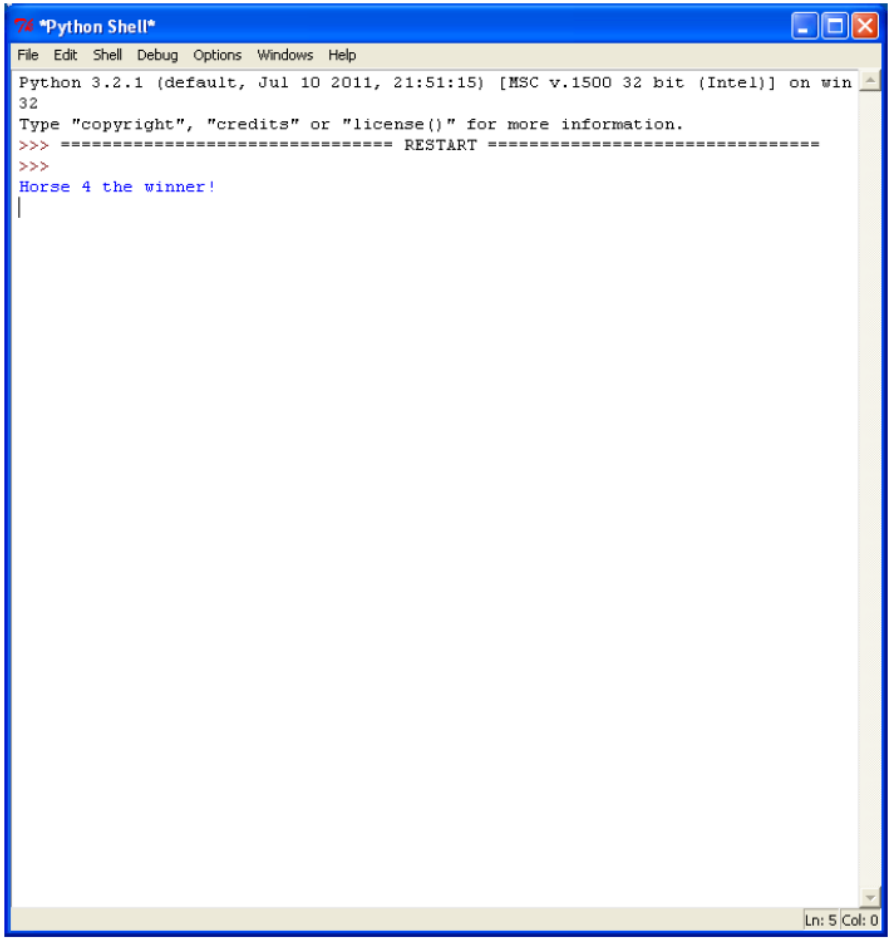
Function `generateHorses` (**lines 26–32**) is implemented the same as in stage 1 (to return a list of turtle objects), except that it is passed a list of horse images, rather than the number of horses to generate. Thus, `generateHorses` in **line 65** is passed the list of images in variable `horse_images`.

Program Implementation

Stage 3 – Animating the Horses

Next we develop and test the program with additional code that animates the horses so that they are randomly advanced until the first horse crosses the finish line. The number of the winning horse is displayed in the Python shell.


Horse Race Simulation Program
10
9
8
7
6
5
4
3
2
1


Python Shell
File Edit Shell Debug Options Windows Help
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!
Ln: 5 Col: 0

```
1  # Horse Racing Program (Stage 3)
2
3  import turtle
4  import random
5
6  def getHorseImages(num_horses):
7      # init empty list
8      images = []
9
10     # get all horse images
11     for k in range(0, num_horses):
12         images = images + ['horse_' + str(k+1) + '_image.gif']
13
14     return images
15
16 def registerHorseImages(images):
17     for k in range(0, len(images)):
18         turtle.register_shape(images[k])
19
20 def newHorse(image_file):
21     horse = turtle.Turtle()
22     horse.hideturtle()
23     horse.shape(image_file)
24
25     return horse
26
27 def generateHorses(images, num_horses):
28     horses = []
29     for k in range(0, num_horses):
30         horse = newHorse(images[k])
31         horses.append(horse)
32
33     return horses
34
35 def placeHorses(horses, loc, separation):
36     for k in range(0, len(horses)):
37         horses[k].hideturtle()
38         horses[k].penup()
39         horses[k].setposition(loc[0], loc[1] + k * separation)
40         horses[k].setheading(180)
41         horses[k].showturtle()
42         horses[h].pendown()
43
44 def startHorses(horses, finish_line, forward_incr):
45     # init
46     have_winner = False
47
48     k = 0
49     while not have_winner:
50         horse = horses[k]
51         horse.forward(random.randint(1, 3) * forward_incr)
52
53         # check for horse over finish line
54         if horse.position()[0] < finish_line:
55             have_winner = True
56         else:
57             k = (k + 1) % len(horses)
58     return k
59
```

Two new functions are added in this version of the program, `startHorses` and `displayWinner`.

Function `startHorses` (**lines 44–58**) is passed the list of horse turtle objects, the location of the finish line, and the fundamental increment amount. Each horse is randomly advanced by one to three times this amount. The while loop for incrementally moving the horses is on **line 48**. The loop iterates until a winner is found (until the `have_winner` is `True`).

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

```
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()
```

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` on **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.

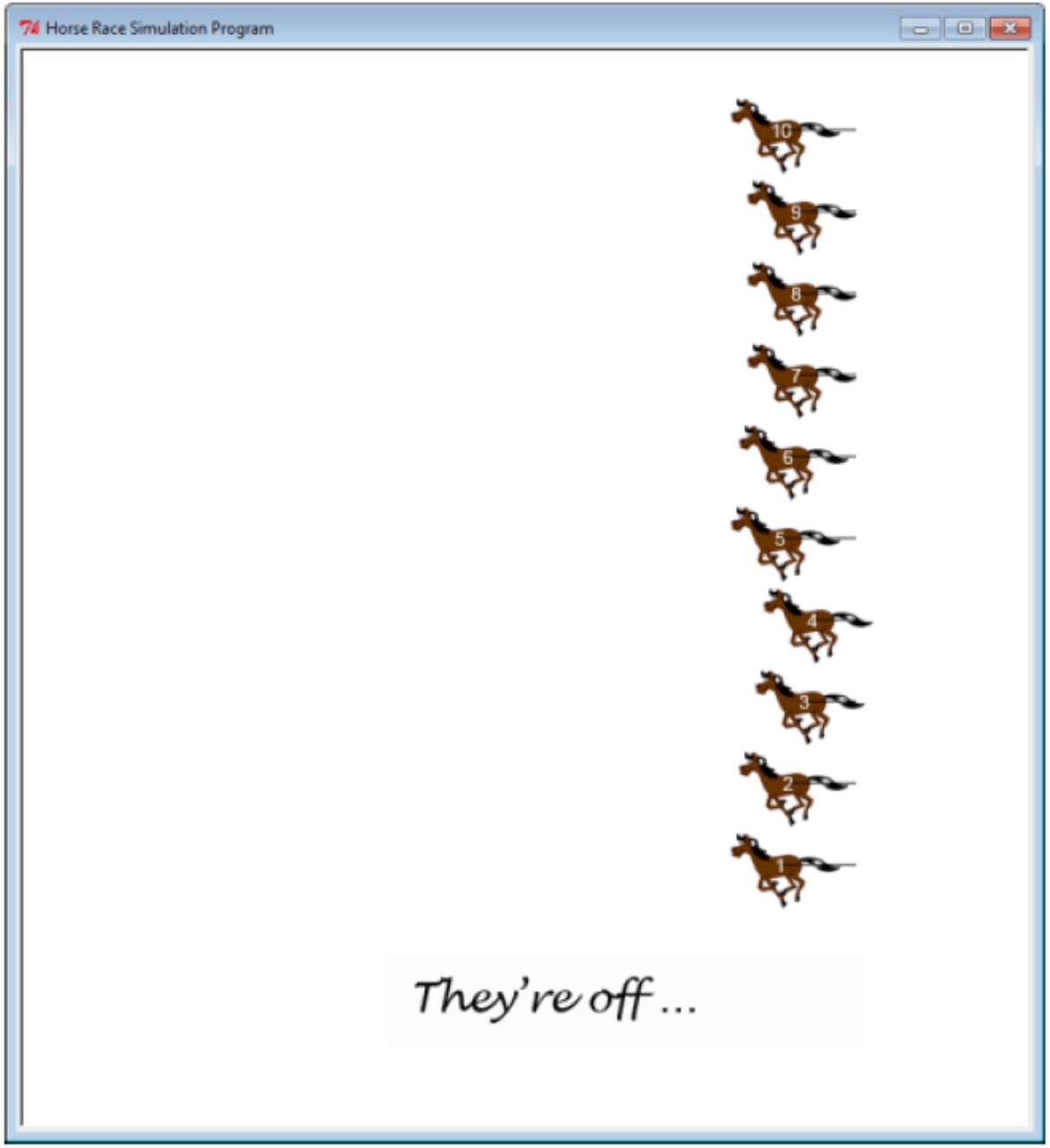
Function `displayWinner` displays the winning horse number in the Python shell (**lines 60–61**). This function will be rewritten in the next stage of program development to display a "winner" banner image in the turtle screen. Thus, this implementation of the function is for testing purposes only.

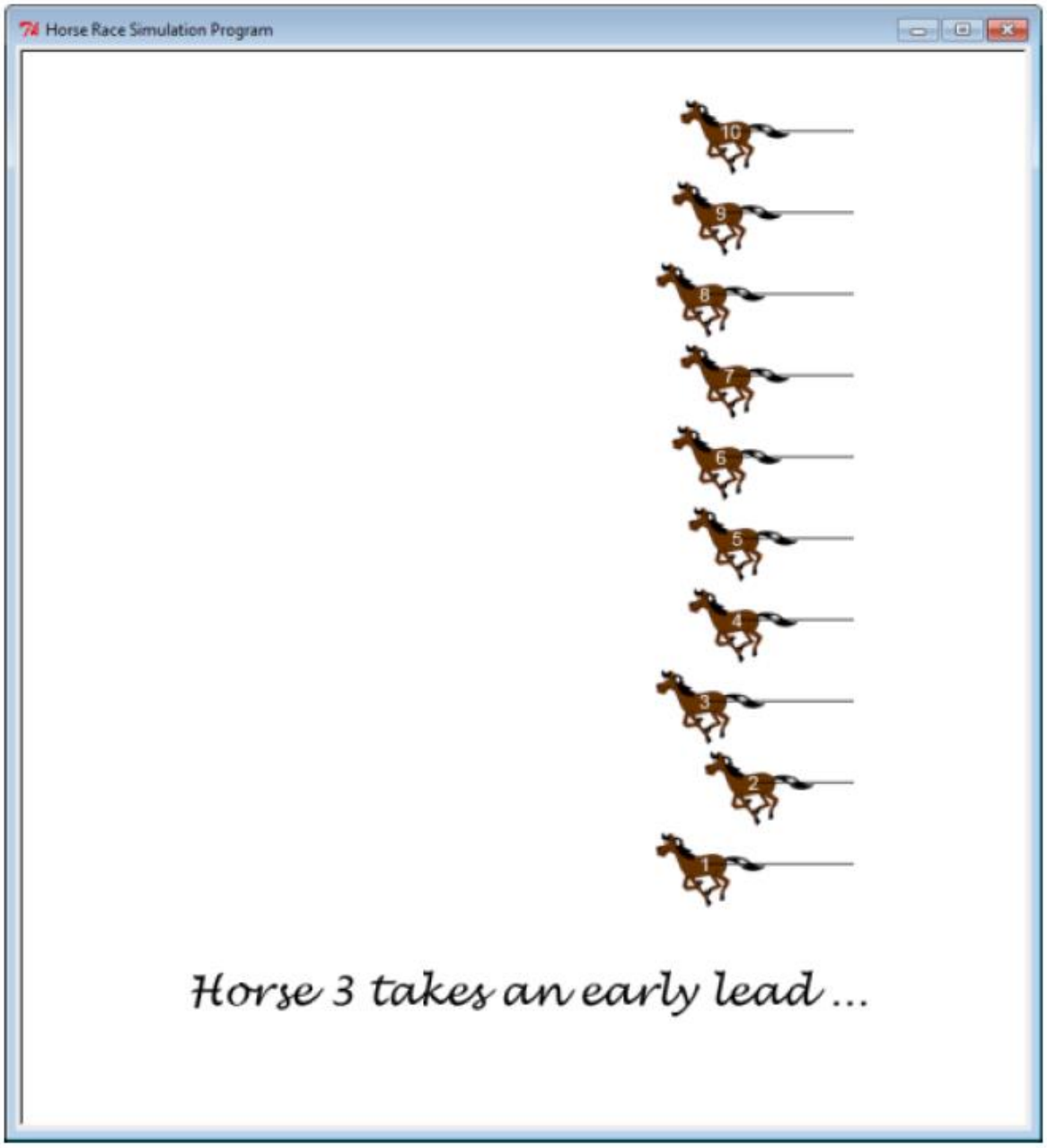
The main program section (**lines 63–100**) is the same as in the previous stage of program development, except for the inclusion of the calls to functions `startHorses` and `displayWinner` on **lines 94** and **97**.

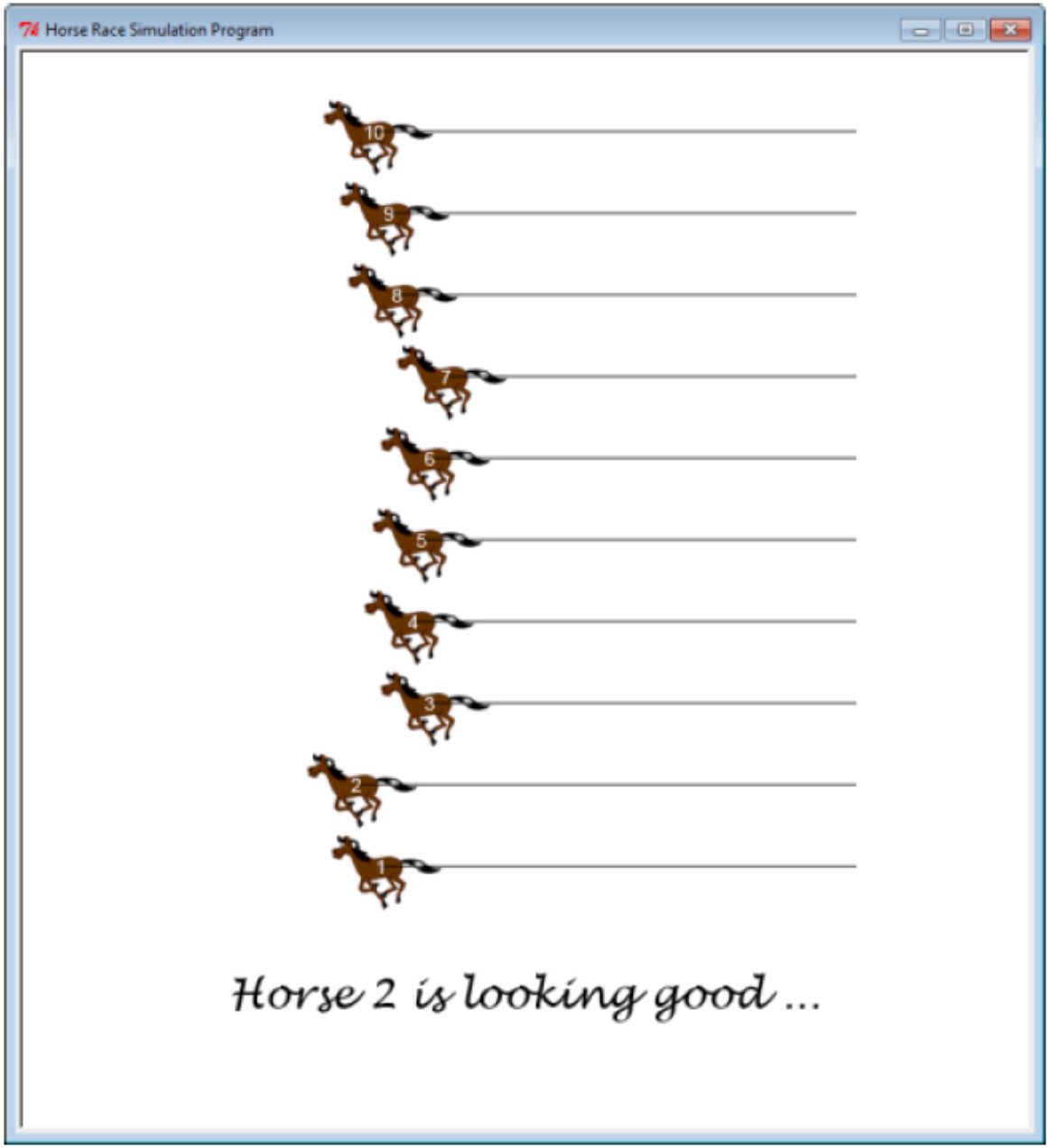
Program Implementation

Final Stage – Adding Race Banners

Finally, we add the code for displaying banners at various points in the race. In addition, the winning horse is made to blink.


Horse Race Simulation Program
10
9
8
7
6
5
4
3
2
1
They're off ...


Horse Race Simulation Program
10
9
8
7
6
5
4
3
2
1
Horse 3 takes an early lead ...


Horse Race Simulation Program
10
9
8
7
6
5
4
3
2
1
Horse 2 is looking good ...


Horse Race Simulation Program
10
9
8
7
6
5
4
3
2
1
We Have a Winner!

```
1  # Horse Racing Program (Final Stage)
2
3  import turtle
4  import random
5  import time
6
7  def getHorseImages(num_horses):
8      # init empty list
9      images = []
10
11     # get all horse images
12     for k in range(0, num_horses):
13         images = images + ['horse_' + str(k + 1) + '_image.gif']
14
15     return images
16
17 def getBannerImages(num_horses):
18     # init empty list
19     all_images = []
20
21     # get "They're Off" banner image
22     images = ['theyre_off_banner.gif']
23     all_images.append(images)
24
25     # get early lead banner images
26     images = []
27     for k in range(0, num_horses):
28         images = images + ['lead_at_start_' + str(k + 1) + '.gif']
29     all_images.append(images)
30
31     # get mid-way lead banner images
32     images = []
33     for k in range(0, num_horses):
34         images = images + ['looking_good_' + str(k + 1) + '.gif']
35     all_images.append(images)
36
37     # get "We Have a Winner" banner image
38     images = ['winner_banner.gif']
39     all_images.append(images)
40
41     return all_images
42
43 def registerHorseImages(images):
44     for k in range(0, len(images)):
45         turtle.register_shape(images[k])
46
```

The final version of the program imports one additional module, Python Standard Library module `time` (**line 5**). The program uses method `sleep` from the `time` module to control the blinking rate of the image of the winning horse.

Function `getBannerImages` (**line 17**), along with functions `registerBannerImages` (**line 47**), and `displayBanner` (**line 84**) incorporate the banner images into the program the same way that the horse images were incorporated in the previous program version.


```
47  def registerBannerImages(images):
48      for k in range(0, len(images)):
49          for j in range(0, len(images[k])):
50              turtle.register_shape(images[k][j])
51
52  def newHorse(image_file):
53      horse = turtle.Turtle()
54      horse.hideturtle()
55      horse.shape(image_file)
56
57      return horse
58
59  def generateHorses(images, num_horses):
60      horses = []
61      for k in range(0, num_horses):
62          horse = newHorse(images[k])
63          horses.append(horse)
64
65      return horses
66
67  def placeHorses(horses, loc, separation):
68      for k in range(0, len(horses)):
69          horses[k].hideturtle()
70          horses[k].penup()
71          horses[k].setposition(loc[0], loc[1] + k * separation)
72          horses[k].setheading(180)
73          horses[k].showturtle()
74          horses[k].pendown()
75
76  def findLeadHorse(horses):
77      # init
78      lead_horse = 0
79
80      for k in range(1, len(horses)):
81          if horses[k].position()[0] <  \
82             horses[lead_horse].position()[0]:
83              lead_horse = k
84      return lead_horse
85
86  def displayBanner(banner, position):
87      the_turtle = turtle.getturtle()
88      the_turtle.setposition(position[0], position[1])
89      the_turtle.shape(banner)
90      the_turtle.stamp()
91
```

Added functions `getBannerImages`, `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. When the banners appear during a race is based on the location of the currently leading horse.

```
92  def startHorses(horses, banners, finish_line, forward_incr):
93      # init
94      have_winner = False
95      early_leading_horse_displayed = False
96      midrace_leading_horse_displayed = False
97
98      # display "They're Off" banner image
99      displayBanner(banner_images[0][0], (70, -300))
100
101     k = 0
102     while not have_winner:
103         horse = horses[k]
104         horse.forward(random.randint(1, 3) * forward_incr)
105
106         # display mid-race lead banner
107         lead_horse = findLeadHorse(horses)
108         if horses[lead_horse].position()[0] < -125 and \
109            not midrace_leading_horse_displayed:
110
111             displayBanner(banners[2][lead_horse], (40, -300))
112             midrace_leading_horse_displayed = True
113
114         # display early lead banner
115         elif horses[lead_horse].position()[0] < 125 and \
116            not early_leading_horse_displayed:
117             displayBanner(banners[1][lead_horse], (10, -300))
118             early_leading_horse_displayed = True
119
120         # check for horse over finish line
121         if horse.position()[0] < finish_line:
122             have_winner = True
123         else:
124             k = (k + 1) % len(horses)
125     return k
126
```

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.

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`.

```
127 def displayWinner(winning_horse, winner_banner):
128     # display "We Have a Winner" banner
129     displayBanner(winner_banner, (20, -300))
130
131     # blink winning horse
132     show = False
133     blink_counter = 5
134     while blink_counter != 0:
135         if show:
136             winning_horse.showturtle()
137             show = False
138             blink_counter = blink_counter - 1
139         else:
140             winning_horse.hideturtle()
141             show = True
142
143         time.sleep(.4)
144
145 # ---- main
146
147 # init number of horses
148 num_horses = 10
149
150 # set window size
151 turtle.setup(750, 800)
152
153 # get turtle window
154 window = turtle.Screen()
155
156 # set window title
157 window.title('Horse Race Simulation Program')
158
159 # hide default turtle and keep from drawing
160 the_turtle.hideturtle()
161 the_turtle.penup()
162
163 # init screen layout parameters
164 start_loc = (240, -200)
165 finish_line = -240
166 track_separation = 60
167 forward_incr = 6
168
169 # register images
170 horse_images = getHorseImages()
171 banner_images = getBannerImages()
172 registerHorseImages(horse_images)
173 registerBannerImages(banner_images)
174
```

This version of `displayWinner` (**line 127**) replaces the previous version that simply displayed the winning horse number in the shell window. A "count-down" variable, `blink_counter`, is set to 5 on **line 133**, decrementing it to zero in the while loop, causing the winning horse to blink five times.

Boolean variable `show`, initialized to False on **line 132** , is used to alternately show and hide the turtle. The `sleep` method, called on **line 143**, causes the program to suspend execution for four-tenths of a second so that the showing/hiding of the winning horse image switch slowly enough to cause a blinking effect.

The default turtle 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 the location where banners are displayed, the default turtle is hidden and its pen attribute is set to "up" (**lines 160–161).**

```
175 # generate and init horses
176 horses = generateHorses(horse_images)
177
178 # place horses at starting line
179 placeHorses(horses, start_loc, track_separation)
180
181 # start horses
182 winner = startHorses(horses, banner_images, finish_line,
183                      forward_incr)
184
185 # light up for winning horse
186 displayWinner(horses[winner], banner_images[3][0])
187
188 # terminate program when close window
189 turtle.exitonclick()
```

The only change in the main module of the program is related to the display of banner images. Added **lines 171** and **173** register and display the banners. The calls to `startHorses` and `displayWinner` on **lines 182** and **186** are changed (and the corresponding function definitions) to each pass one more argument consisting of banner images.