Lone Star
section 4

Classes Part 3 quiz

1. (13.1) Write a class for a **Point** with the instance variables and methods listed below.
   A point in the coordinate plane has an x and y coordinate. Points can be compared, and the distance between two points can be calculated using the distance formula.

   For example:

   - $p_1 = (3, 4)$
   - $p_2 = (0, 0)$
   - Distance between p_1 and p_2 is $\sqrt{(3-0)^2 + (4-0)^2} = 5$
   - $(3, 4)$ is equal to $(3, 4)$, but $(1, 2)$ is not equal to $(6, 6)$.

Point
x (x-coordinate)
y (y-coordinate)
__init__
__eq__
distance(other)
__str__

   Your class should support:

   - Creating a point with x and y coordinates
   - Determine if two points are equal using the __eq__ method
   - Calculating the distance to another point using `distance(other)`
   - Printing a readable version of the point

   Once you have created the class, add code that:

   - Instantiate two Points
   - Compare if they are equal
   - Print a readable version of one of the Points you created.

2. (13.2) Write a class for a **Time** object with the instance variables and methods listed below.
   Time can be represented in hours and minutes (e.g., 2 hours and 45 minutes).
   When adding times, make sure minutes are correctly carried into hours.

   For example:

   - $t_1 = 1$ hour 30 minutes
   - $t_2 = 2$ hours 45 minutes
   - $t_1 + t_2 = 4$ hours 15 minutes (because $30 + 45 = 75$ minutes, which becomes 1 hour 15 minutes)

Time
hours
minutes
__init__
__add__
__str__

   Your class should support:

   - Creating a time object with hours and minutes
   - Adding two times using the __add__ method
   - Printing time in a readable format (e.g., "2h 45m")
     Hint: you don't need to consider days. You can have 30 hours.

   Once you have created the class, add code that:

   - Creates two time objects
   - Adds them together
   - Prints the result

3. (13.3) Write a class for a **Playlist** with the instance variables and methods listed below.
A Playlist should have a default name of "New Playlist".
It can be instantiated with initial songs, but it is not required to.
Create a method called *add_song* which adds a song title (a string) to the Playlist.
You should be able to combine two Playlists, and print them in a readable way.

For example:

- p_1 = ["Song A", "Song B"]
- p_2 = ["Song C"]
- $p_1 + p_2$ = ["Song A", "Song B", "Song C"]

Playlist
name
songs (list of strings)
__init__
add_song
__add__
__str__

Your class should support:

- Creating a playlist with a name and list of songs
- Adding two playlists (combines song lists)
- Printing the playlist in a readable way (e.g., list songs)

Once you have created the class, add code that:

- Creates two playlists and at least one song to each.
- Combines the playlists
- Prints the result

1. (13.1) Write a class for a **Vector** with the instance variables and methods listed below.
   A vector in the xy-plane is a quantity that has both direction and magnitude.
   It can be written in the form $v = ax + by$, where a and b are real numbers.

   Some examples of vectors include:

   - $v_1 = 3x + 2y$
   - $v_2 = -2x + 6y$
   - $v_3 = 2x$ (which is the same as $2x + 0y$)

Vector
a (x-component) b (y-component)
__init__ __eq__ __str__

   Your class should support:

   - Creating a vector with x and y components
   - Comparing if two vectors are equal using the __eq__ method
   - Printing a readable version of the vector

   Hint: Two vectors are equal if their components are equal. That is, the x-components of both are equal and the y-components of both are equal.
   For example, $v_1 = (2x + 3y)$ and $v_2 = (2x + 3y)$ are equal,
   but $v_1 = (2x + 3y)$ and $v_2 = (4x + 5y)$ are not.

   After writing the class, initialize three vectors and write code to add them together.

2. (13.2) Write a class for an **RGBColor** with the instance variables and methods listed below. Colors on a screen are often represented using Red, Green, and Blue components (values from 0 to 255). Colors can be added together by taking the average of adding each component.

   Some examples of RGB colors:

   - $c_1 = (170,\ 150,\ 200)$
   - $c_2 = (30,\ 100,\ 60)$
   - $c_3 = c_1 + c_2 = (\frac{170+30}{2},\ \frac{150+100}{2},\ \frac{200+60}{2})$ becomes $(100,\ 125,\ 130)$

RGBColor
r (red) g (green) b (blue)
__init__ __add__ __str__

   Your class should support:

   - Creating a color with red, green, and blue values (each from 0 to 255)
   - Adding two colors using the __add__ method (cap each result at 255)
   - Printing the color in a readable format (e.g.,"RGB(150, 200, 255)")

   After writing the class, create three colors and write code to add them together. Print the result.

3. (13.3) Write a class for a **Playlist** with the instance variables and methods listed below.
   A Playlist should have a default name of "New Playlist".
   It can be instantiated with initial songs, but it is not required to.
   Create a method called *add_song* which adds a song title (a string) to the Playlist.
   You should be able to combine two Playlists, and print them in a readable way.

For example:

- $p_1 =$ [“Song A”, “Song B”]
- $p_2 =$ [“Song C”]
- $p_1 + p_2 =$ [“Song A”, “Song B”, “Song C”]

Playlist
name songs (list of strings)
__init__ add_song __add__ __str__

Your class should support:

- Creating a playlist with a name and list of songs
- Adding two playlists (combines song lists)
- Printing the playlist in a readable way (e.g., list songs)

Once you have created the class, add code that:

- Creates two playlists and at least one song to each.
- Combines the playlists
- Prints the result

Dark Helmet
section 4

Classes Part 3 quiz

1. (13.1) Write a class for a **ComplexNumber** with the instance variables and methods listed below.
A ComplexNumber is a number of the form $a + bi$, where a is the real part, b is the imaginary part, and both a and b are real numbers.

   Some examples of a ComplexNumber include:

   - $z_1 = 3 + 2i$
   - $z_2 = -1 + 4i$
   - $z_3 = 2$ (which is the same as $2 + 0i$)

ComplexNumber
a (real part) b (imaginary part)
__init__ __eq__ __str__

   Once you have created the class, add code that:

   - Instantiates two ComplexNumbers with a real and imaginary parts
   - Compares if two ComplexNumbers are equal using the __eq__ method
   - Printing a readable version (e.g., "3 + 2i" or "5 - 1i")

   After writing the class, initialize two ComplexNumbers and write code to determine if they are equal.

2. (13.2) Write a class for a **LinearEquation** with the instance variables and methods listed below.
A linear equation is of the form $y = mx + b$, where m is the slope and b is the y-intercept.

   For example:

   - $y_1 = 2x + 3$
   - $y_2 = -x + 5$

LinearEquation
m (slope) b (y-intercept)
__init__ __add__ __str__

   You should be able to add two equations together:

   - $y_1 + y_2 = (2 - 1)x + (3 + 5) = x + 8$

   Your class should support:

   - Creating a linear equation with slope and y-intercept
   - Adding two equations using the __add__ method
   - Printing a readable version of the equation

   Once you have created the class, add code that:

   - Instantiate two linear equations
   - Add them together
   - print a readable version of one of the LinearEquations you created.

3. (13.3) Write a class for a **Playlist** with the instance variables and methods listed below.
A Playlist should have a default name of "New Playlist".
It can be instantiated with initial songs, but it is not required to.
Create a method called *add_song* which adds a song title (a string) to the Playlist.
You should be able to combine two Playlists, and print them in a readable way.

For example:

Playlist
name songs (list of strings)
__init__ add_song __add__ __str__

- $p_1 =$ ["Song A", "Song B"]
- $p_2 =$ ["Song C"]
- $p_1 + p_2 =$ ["Song A", "Song B", "Song C"]

Your class should support:

- Creating a playlist with a name and list of songs
- Adding two playlists (combines song lists)
- Printing the playlist in a readable way (e.g., list songs)

Once you have created the class, add code that:

- Creates two playlists and at least one song to each.
- Combines the playlists
- Prints the result

President Skroob
section 1

Classes Part 3 quiz

1. (13.1) Write a class for a **Point** with the instance variables and methods listed below.
   A point in the coordinate plane has an x and y coordinate. Points can be compared, and the distance between two points can be calculated using the distance formula.

   For example:

Point
x (x-coordinate)
y (y-coordinate)
__init__
__eq__
distance(other)
__str__

   - $p_1 = (3, 4)$
   - $p_2 = (0, 0)$
   - Distance between p_1 and p_2 is $\sqrt{(3-0)^2 + (4-0)^2} = 5$
   - $(3, 4)$ is equal to $(3, 4)$, but $(1, 2)$ is not equal to $(6, 6)$.

   Your class should support:

   - Creating a point with x and y coordinates
   - Determine if two points are equal using the __eq__ method
   - Calculating the distance to another point using `distance(other)`
   - Printing a readable version of the point

   Once you have created the class, add code that:

   - Instantiate two Points
   - Compare if they are equal
   - Print a readable version of one of the Points you created.

2. (13.2) Write a class for a **Time** object with the instance variables and methods listed below.
   Time can be represented in hours and minutes (e.g., 2 hours and 45 minutes).
   When adding times, make sure minutes are correctly carried into hours.

   For example:

Time
hours
minutes
__init__
__add__
__str__

   - $t_1 = 1$ hour 30 minutes
   - $t_2 = 2$ hours 45 minutes
   - $t_1 + t_2 = 4$ hours 15 minutes (because $30 + 45 = 75$ minutes, which becomes 1 hour 15 minutes)

   Your class should support:

   - Creating a time object with hours and minutes
   - Adding two times using the __add__ method
   - Printing time in a readable format (e.g., "2h 45m")
     Hint: you don't need to consider days. You can have 30 hours.

   Once you have created the class, add code that:

   - Creates two time objects
   - Adds them together
   - Prints the result

3. (13.3) Write a class for a **ShoppingCart** with the instance variables and methods listed below.
It can be instantiated with initial items in the cart, but it is not required to.
Create a method called *add_items* which adds an item (a string) to the ShoppingCart. The same item can be added more than once. If the item is already in the ShoppingCart, increase its quantity by one. If its not in the ShoppingCart, set the quantity to 1.
Hint: think about what type of data structure you should use.
When two ShoppingCarts are added together, the result should be a ShoppingCart containing the items in both. Overlapping items should have a sum of their quantities.

For example:

ShoppingCart
items (dict→str:int)
__init__ add_item __add__ __str__

- $p_1 = \{\text{“tea”} : 1, \text{“energy drink”} : 2\}$
- $p_2 = \{\text{“energy drink”} : 3, \text{“hat”} : 1, \}$
- $p_1 + p_2 = \{\text{“tea”} : 1, \text{“energy drink”} : 5, \text{“hat”} : 1, \}$

Your class should support:

- Creating a ShoppingCart
- Adding two ShoppingCarts (combines items)
- Printing the ShoppingCart in a readable way (e.g., lists all items with quantities)

Once you have created the class, add code that:

- Creates two ShoppingCarts and at least one item to each.
- Combines the ShoppingCarts
- Prints the result