Phenotype Leaf Angles of Corn Seedlings Using Computational Methods

In this exercise, students will learn how to use the image analysis program, PlantCV, to analyze images of corn seedlings and provide data on plant characteristics that scientists will use for their research.

Step 1. What do you need?

Computer
Internet
Images of corn seedlings

Step 2. Launch the program PlantCV

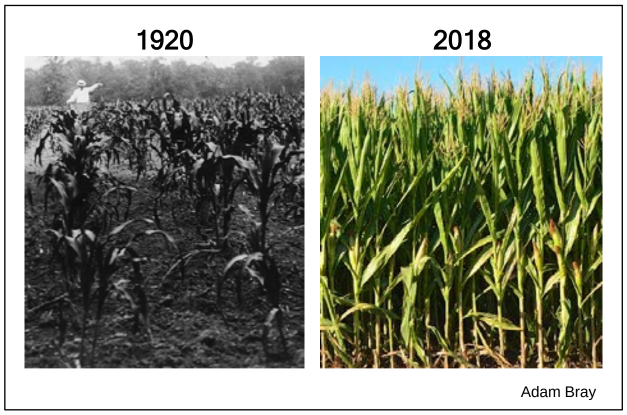
Click on the following link to launch the program. It might take some time to open the interactive activity. You can continue with Step 3 while you wait.

https://mybinder.org/v2/gh/danforthcenter/Eveland_NSF_Outreach/master?filepath=index.ipynb

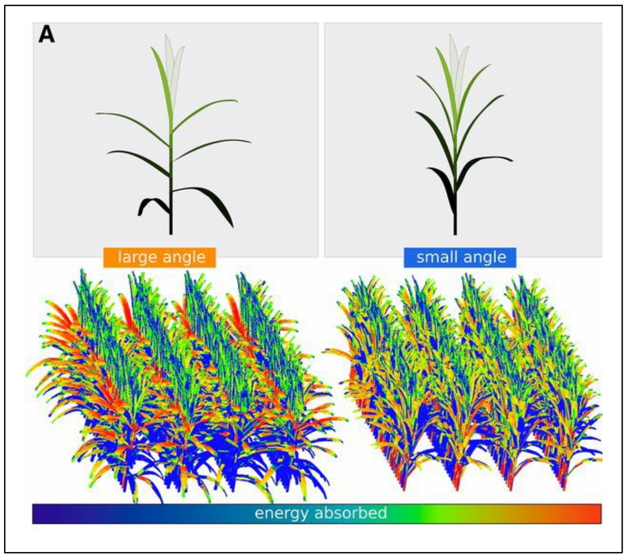
Step 3. Measuring leaf angle in corn

The *phenotype* is the set of observable characteristics of an organism resulting from the interaction of its genotype with the environment. The *genotype* is the set of genes that each organism has. The *environment* involves the conditions in which an organism operates (e.g. diet, stress, temperature, solar radiation, etc.). In plants, the phenotype consists of characteristics such as plant height, biomass, number of flowers, etc. In corn (maize) plants, leaf angle is a phenotypic characteristic that plays an important role in determining plant density (how many plants can be grown per acre) and yield (the number of ears of corn that are produced per acre). Scientists are trying to understand what causes certain genotypes of corn plants to have ideal leaf angles; they can use this information to help farmers pick corn of genotypes which will maximize their planting density and yield.

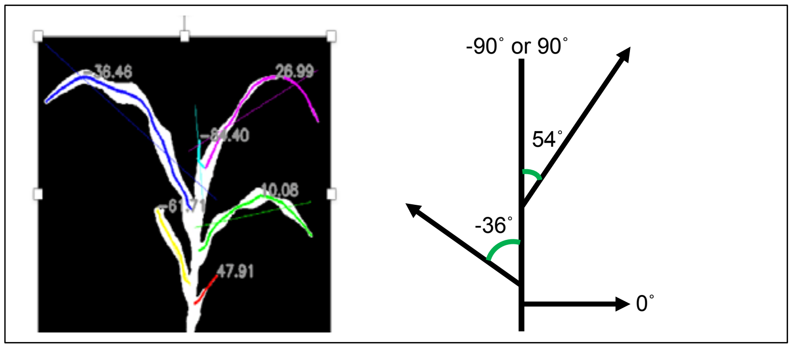
Over the years, scientists have improved corn plants to have more erected leaf angles. The photo below shows the result of this process on the efficiency of a corn field over the course of the past century.



The figure below shows that under dense-planting conditions, a more erect leaf angle enhances plant light interception in the lower canopy, and overall photosynthetic capacity, hence the ability to grow optimally.

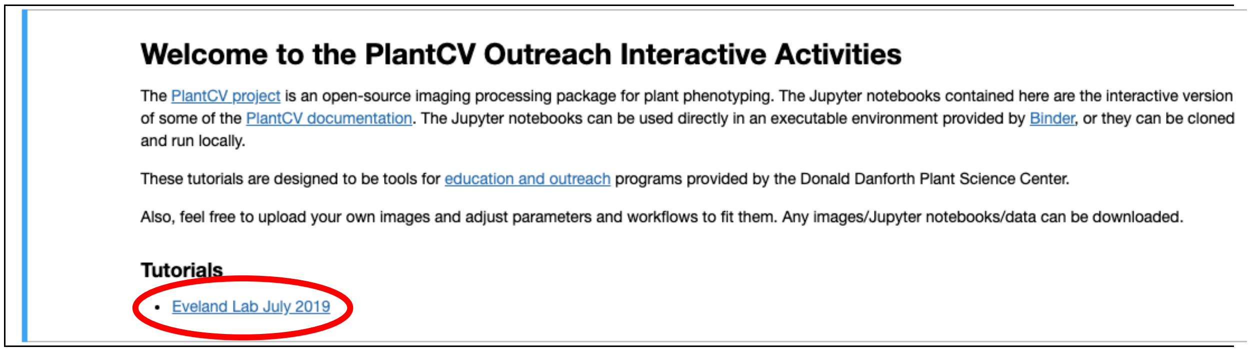


PlantCV is a free image analysis software package made to phenotype plants using computer vision, a form of artificial intelligence that trains computers to interpret visual information like images and videos. You will use PlantCV to analyze images of corn seedlings to computationally measure leaf angle, a phenotypic parameter (see figure below). Scientists will then compare your automated measurements to manual measurements of the leaf angles of the seedling in your image. Manual measurements are often used to “ground truth” computational phenotyping to assess accuracy.



Step 4. Analyze a sample image

1. Go to the screen where PlantCV has been launched.
2. Under the Tutorials heading, click the hyperlink labelled Eveland Lab July 2019 (under the Tutorials heading – indicated with a red oval in the screenshot below). This will direct you to a tutorial that demonstrates how to use PlantCV to measure the leaf angle of a corn plant from a sample image.



Welcome to the PlantCV Outreach Interactive Activities

The PlantCV project is an open-source imaging processing package for plant phenotyping. The Jupyter notebooks contained here are the interactive version of some of the PlantCV documentation. The Jupyter notebooks can be used directly in an executable environment provided by Binder, or they can be cloned and run locally.

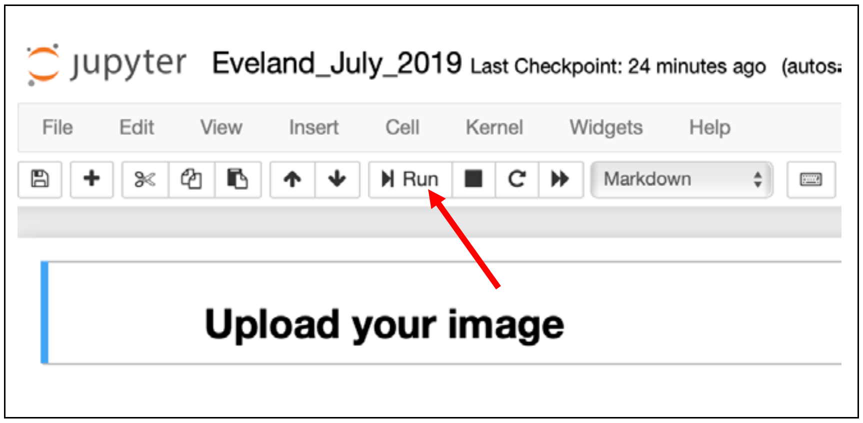
These tutorials are designed to be tools for education and outreach programs provided by the Donald Danforth Plant Science Center.

Also, feel free to upload your own images and adjust parameters and workflows to fit them. Any images/Jupyter notebooks/data can be downloaded.

Tutorials

- Eveland Lab July 2019

3. Under the **Test on the example image** and **How to read code** headings are the instructions to understand how to read and interact with the computer code that controls PlantCV. These instructions are reiterated below:
   3.1. The code is divided into small sections called blocks or lines, indicated by the light grey boxes surrounding each section of code.
   3.2. Within each block of code, there is light green, italicized text (behind a hashtag [#]). This text is a comment - the computer doesn't read this comment; it is a note for the humans reading the code! Read through all comments in all blocks to help understand what each step of code is doing.
   3.3. Although many of the blocks of code will NOT need updating, some parameters may need adjustment and will have little arrows ^ pointing to and describing how to adjust them in the comments.
   3.4. For each block of code, you will select the block by clicking it (a green border should appear around the block), then click the **Run** button (indicated by the red arrow in the screenshot below), left of center on the bar at the top of the activity.

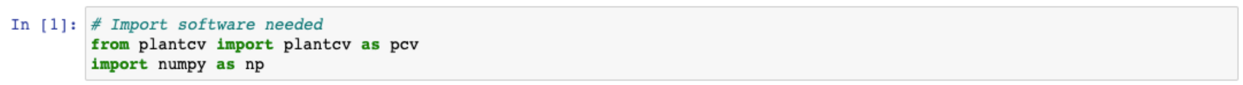


Upload your image

   3.5. While each block of code is being carried out, you will see the label "In [*]" (an abbreviated form of the word 'line,' referring to the line or block of code in question) to the left of the block; when each step is done you will see the * replaced with a number (ex. "In [1]"). **Wait** to see the number in the parenthesis before running the next block.
   3.6. It might take a few seconds to run through the code in each block. Most steps will also plot out an image; these images will appear beneath each block of code when the block is complete.

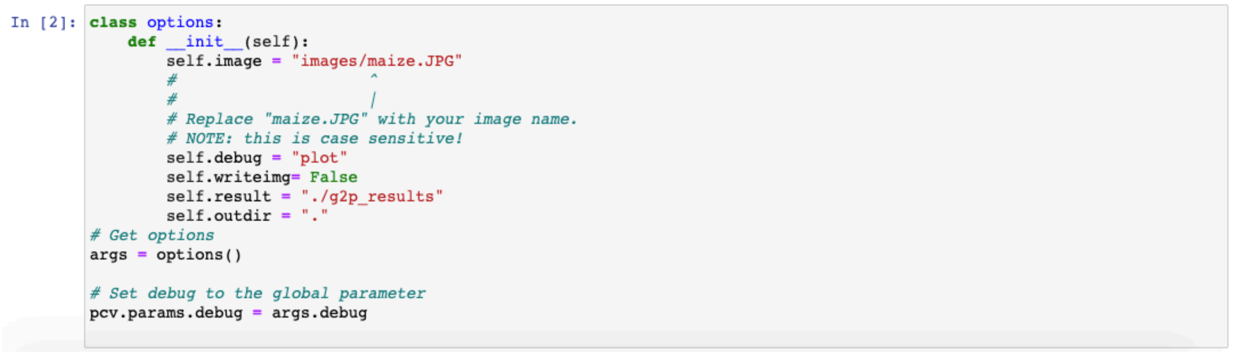
4. To analyze the sample image run each block of code. Use the screenshots below to guide you through each block and make sure your results match the ones shown. It might take a few seconds to run through the code within each block.
5. The numbers of the blocks below may change in your own screen if you need to run block(s) more than once. This will not alter the results.
6. **If you need help**. Share your questions with the PlantCV community by email (plantcv@danforthcenter.org) or through GitHub Danforth PlantCV Issues (https://github.com/danforthcenter/plantcv/issues). You need to create an account to access GitHub.

Block 1 imports the PlantCV software so that it can be run locally on your computer.



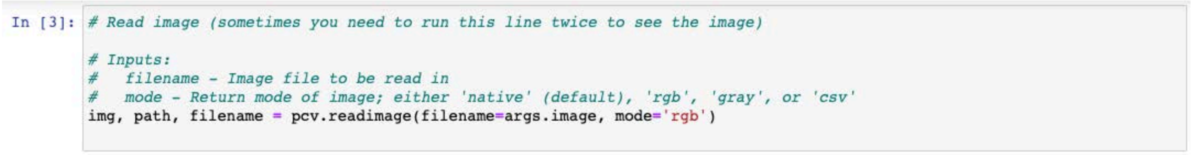
```
In [1]: # Import software needed
        from plantcv import plantcv as pcv
        import numpy as np
```

Block 2 retrieves the image you want to analyze. “images/maize.JPG” is the file pathway and file name of the sample image. This name changes depending on the image being analyzed. Avoid images of plants with yellow tissue since the program does not recognize this color as part of the plant tissue.

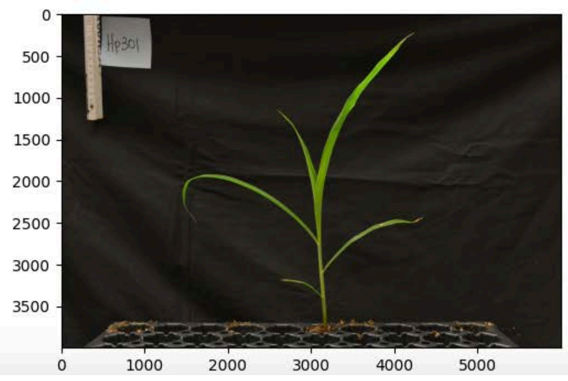


```
In [2]: class options:
            def __init__(self):
                self.image = "images/maize.JPG"
                #                     ^
                #                     |
                # Replace "maize.JPG" with your image name.
                # NOTE: this is case sensitive!
                self.debug = "plot"
                self.writeimg= False
                self.result = "./g2p_results"
                self.outdir = "."
        # Get options
        args = options()

        # Set debug to the global parameter
        pcv.params.debug = args.debug
```

Block 3 displays your image. You should see an image of a corn seedling below the block after this step has finished running.

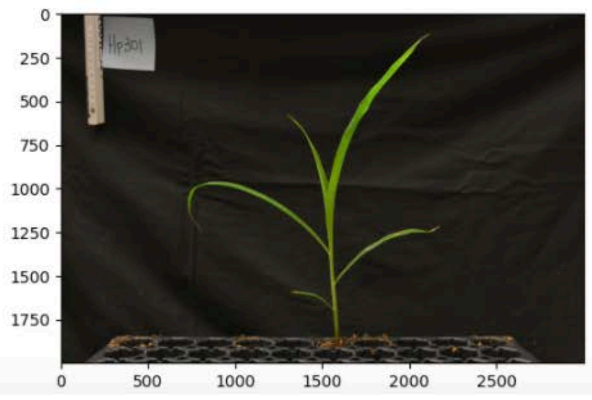


```
In [3]: # Read image (sometimes you need to run this line twice to see the image)

        # Inputs:
        #   filename - Image file to be read in
        #   mode - Return mode of image; either 'native' (default), 'rgb', 'gray', or 'csv'
        img, path, filename = pcv.readimage(filename=args.image, mode='rgb')
```



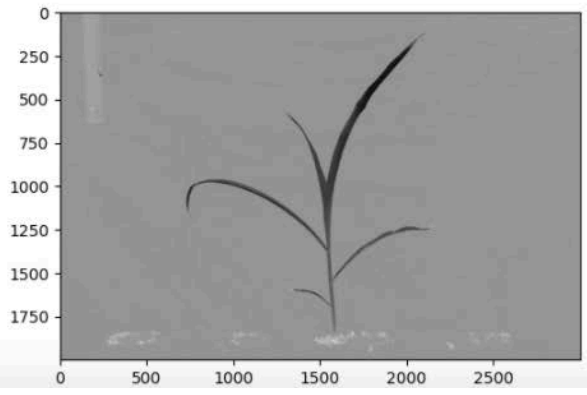
Block 4 resizes the image file so the computer can analyze it quickly. After this block has finished running, you will see the unit scale in the resulting image has decreased by half (compared to the image result of the previous block).

```
In [4]: # The image is quite large which can slow down computation, so resize. The image should look the same but the scale
        # (the x- and y-axis numbers) will be smaller.

        # Inputs:
        #   img - RGB or grayscale image
        #   resize_x - How much to resize in the x axis
        #   resize_y - How much to resize in the y axis
        img = pcv.resize(img=img, resize_x=.5, resize_y=.5)
```

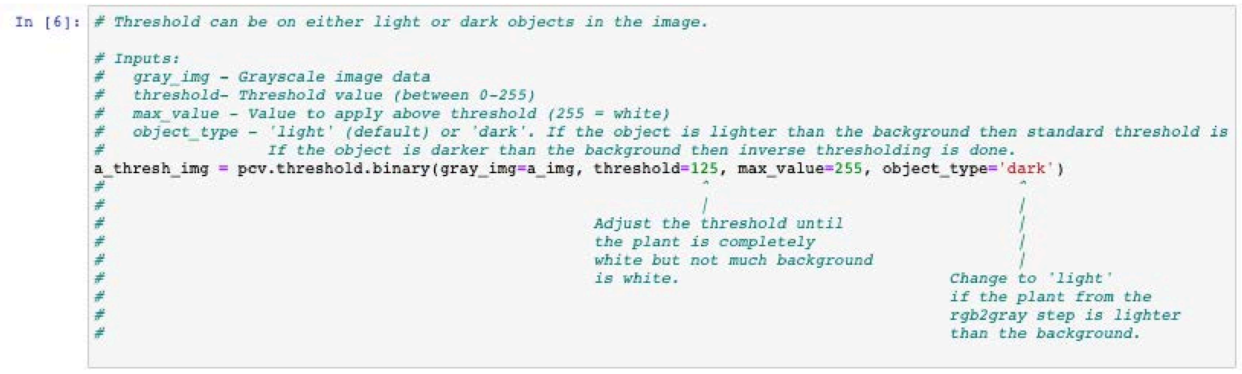


Block 5 allows you to alter the light channel being extracted for image analysis. You can choose to extract channel ‘a’ or ‘b’ in order to make the plant appear much lighter or darker than its surroundings. For this exercise, we can extract channel ‘a’.

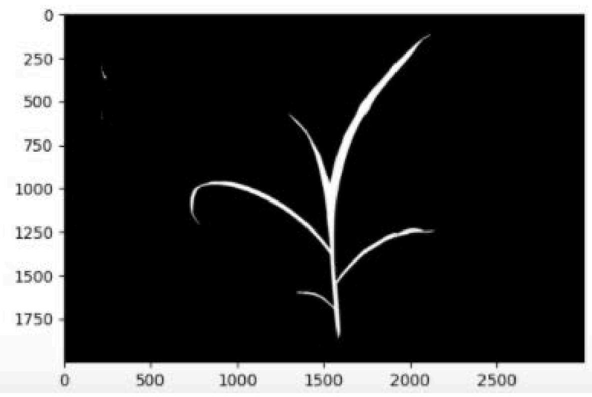
```
In [5]: # Convert RGB to LAB and extract the green-magenta channel ('a')

        # Input:
        #   rgb_img - RGB image data
        #   channel- Split by 'l' (lightness), 'a' (green-magenta), or 'b' (blue-yellow) channel
        a_img = pcv.rgb2gray_lab(rgb_img=img, channel='a')
        #                                               ^
        #                                               |
        #                                    Try changing the channel. We want the plant to be either lighter or darker
        #                                    than the background.
```



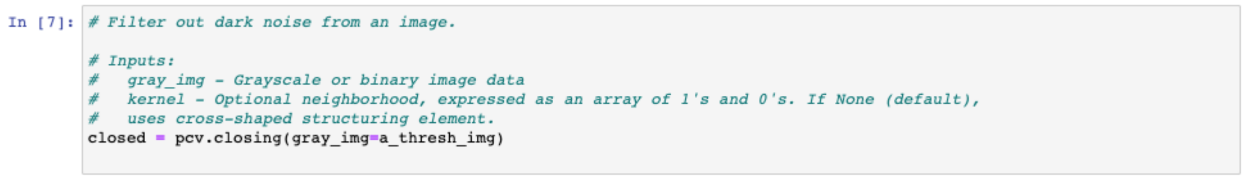
Block 6 helps to identify the seedling in the image by isolating objects much lighter or darker than the background. Because the extracted channel in Block 5 is set to ‘a’, making our plant appear darker than the background, we want to isolate all objects that are **darker** than the background, so we leave the **object type** parameter set to ‘dark’.



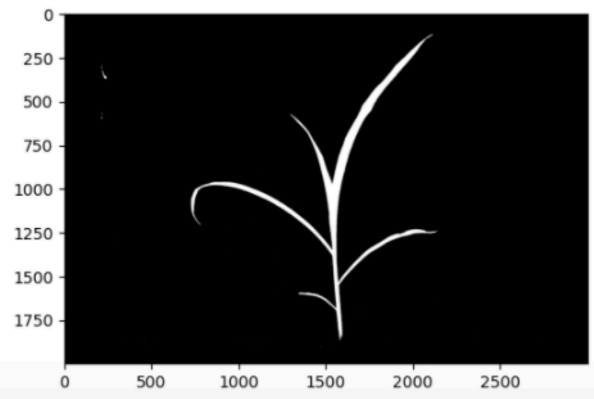
```
In [6]: # Threshold can be on either light or dark objects in the image.

        # Inputs:
        #   gray_img - Grayscale image data
        #   threshold- Threshold value (between 0-255)
        #   max_value - Value to apply above threshold (255 = white)
        #   object_type - 'light' (default) or 'dark'. If the object is lighter than the background then standard threshold is
        #                 If the object is darker than the background then inverse thresholding is done.
        a_thresh_img = pcv.threshold.binary(gray_img=a_img, threshold=125, max_value=255, object_type='dark')
        #                                                               ^                                ^
        #                                                               |                                |
        #                                                   Adjust the threshold until                   |
        #                                                   the plant is completely                      |
        #                                                   white but not much background                |
        #                                                   is white.                             Change to 'light'
        #                                                                                         if the plant from the
        #                                                                                         rgb2gray step is lighter
        #                                                                                         than the background.
```



Block 7 ‘cleans’ the image by removing unnecessary objects from the image.

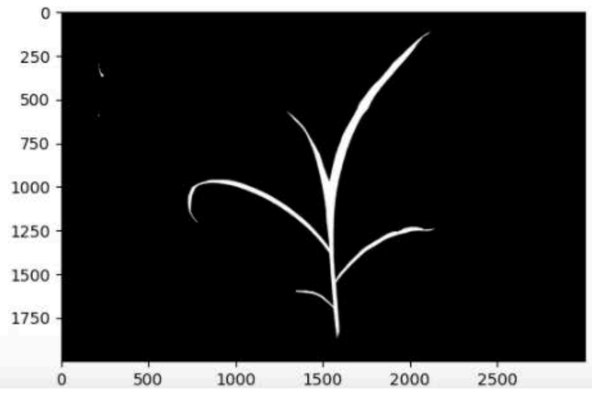


```
In [7]: # Filter out dark noise from an image.

        # Inputs:
        #   gray_img - Grayscale or binary image data
        #   kernel - Optional neighborhood, expressed as an array of 1's and 0's. If None (default),
        #   uses cross-shaped structuring element.
        closed = pcv.closing(gray_img=a_thresh_img)
```



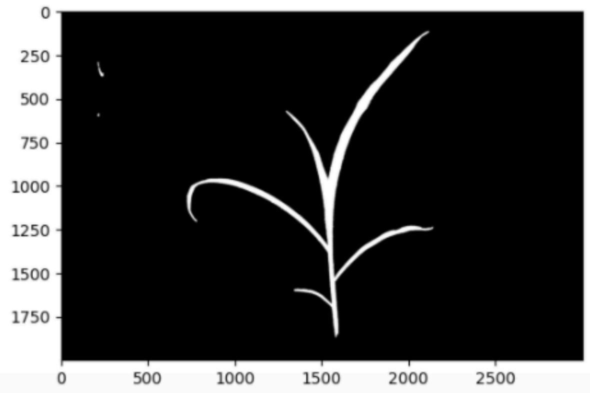
Block 8 further removes unnecessary objects from the image.

```
In [8]: # Fill small objects (reduce image noise)

        # Inputs:
        #   bin_img - Binary image data
        #   size - Minimum object area size in pixels (must be an integer), and smaller objects will be filled
        filled = pcv.fill(bin_img=closed, size=40)
        #                                        ^
        #                                        |
        #                            Increase the size to remove extra things in the background
        #                            but try not to lose any pieces of plant.
```



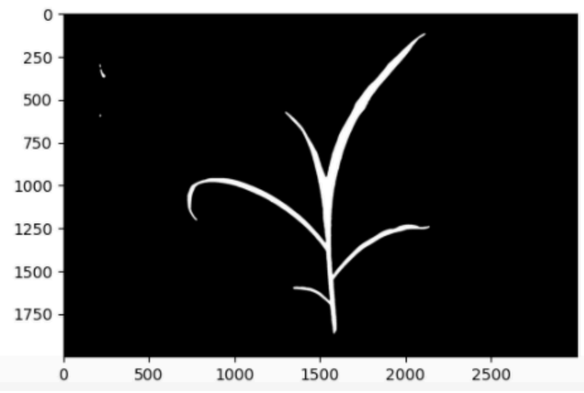
Block 9 helps preserve the plant structure through image alterations and analysis.

```
In [9]: # Dilate the mask to avoid losing leaf tips

        # Inputs:
        #    gray_img = input image
        #    ksize    = kernel size, integer
        #    i        = iterations, i.e. number of consecutive filtering passes
        dilated = pcv.dilate(gray_img=filled, ksize=5, i=1)
```



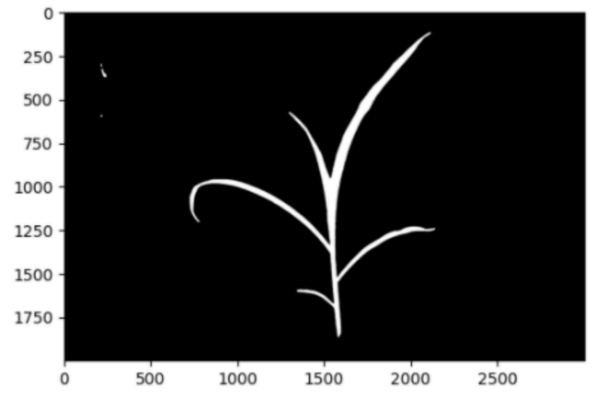
Block 10 smooths lines on the image to reduce “noise” in the analysis.

```
In [10]: # Use a lowpass (blurring) filter to smooth sobel image

         # Inputs:
         #   gray_img - Grayscale image data
         #   ksize - Kernel size (integer or tuple), (ksize, ksize) box if integer input,
         #           (n, m) box if tuple input
         m_blur = pcv.median_blur(gray_img=dilated, ksize=12)
```



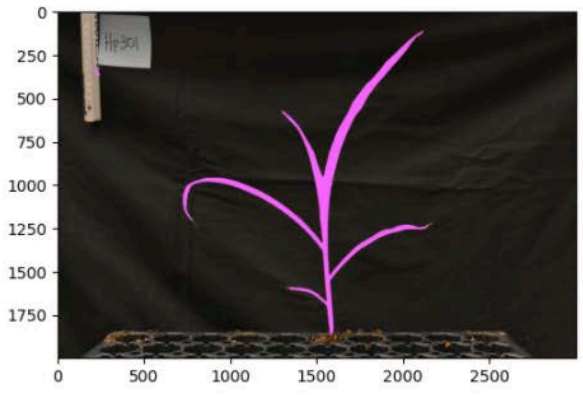
Block 11 fills any holes that may exist in the plant structure outline.

```
In [11]: # Fill in any holes in the plant mask

         # Inputs:
         #   bin_img - Binary image data
         filled_mask = pcv.fill_holes(bin_img=m_blur)
```



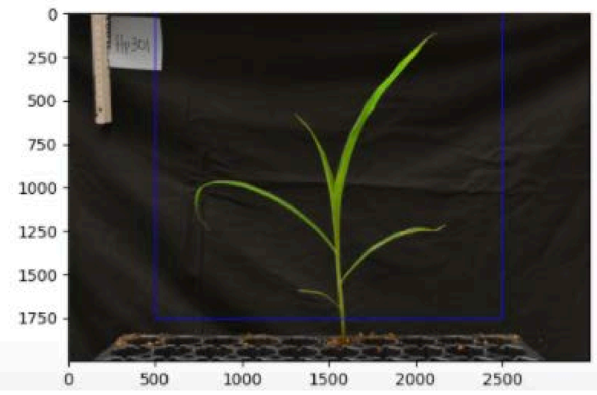
Block 12 identifies the objects that have been located in the image through the previous blocks.

```
In [12]: # Identify objects

         # Inputs:
         #   img - RGB or grayscale image data for plotting
         #   mask - Binary mask used for detecting contours
         obj_cnt, obj_hierarchy = pcv.find_objects(img=img, mask=filled_mask)
```



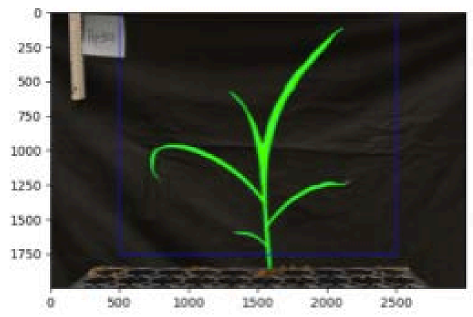
Block 13 defines a region of interest (ROI), which is where analysis of the image will take place. You want the region of interest to include the entire plant, but no other objects. If the image has a leaf that is separated from the plant, this leaf is not considered part of the plant but a separate polygon. Thus, this leaf should be deleted from the image or avoided when defining the ROI. The x, y, height (h), and width (w) values below will work for the sample image. You need to adjust those values for other images.

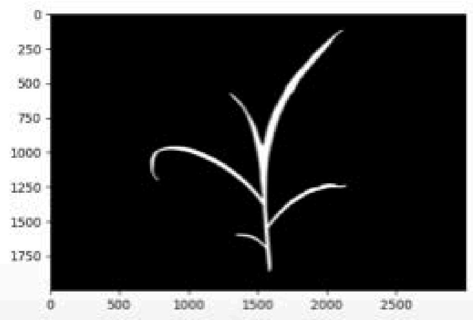
```
In [13]: # Define region of interest (ROI)

         # Inputs:
         #   img - RGB or grayscale image to plot the ROI on
         #   x - The x-coordinate of the upper left corner of the rectangle
         #   y - The y-coordinate of the upper left corner of the rectangle
         #   h - The height of the rectangle
         #   w - The width of the rectangle
         roi_cnt, roi_hierarchy = pcv.roi.rectangle(img=img, x=500, y=0, h=1760, w=2000)
         #                                                       ^                            ^
         #                                                       |                            |
         #                                                       ------------------------------
         #                                                                Update these

         # Depending on the resolution of an image, this will likely need updating.
         # We want the rectangle to at least partially contain the plant but avoids
         # any large background objects that have yet to be filtered out.
```



Block 14 completely removes all unnecessary objects from the image.

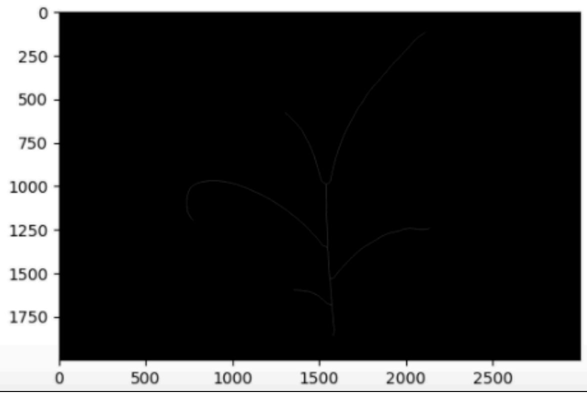
```
In [14]: # Decide which objects to keep

         # Inputs:
         #    img            = img to display kept objects
         #    roi_contour    = contour of roi, output from any ROI function
         #    roi_hierarchy  = contour of roi, output from any ROI function
         #    object_contour = contours of objects, output from pcv.find_objects function
         #    obj_hierarchy  = hierarchy of objects, output from pcv.find_objects function
         #    roi_type       = 'partial' (default, for partially inside), 'cutto', or
         #    'largest' (keep only largest contour)
         plant_obj, plant_hier, plant_mask, obj_area = pcv.roi_objects(img=img, roi_contour=roi_cnt
                                                                       roi_hierarchy=roi_hierarchy
                                                                       object_contour=obj_cnt,
                                                                       obj_hierarchy=obj_hierarchy
                                                                       roi_type='partial')
```





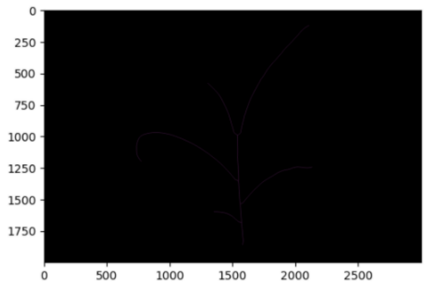
Block 15 creates a one pixel-wide skeleton of the plant structure. In this image is hard to see.

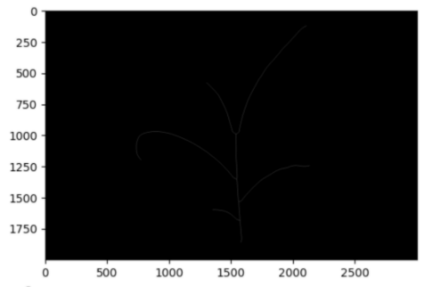
```
In [15]: # Skeletonize the plant mask (one-pixel wide representation)

         # Inputs:
         #   mask - Binary mask
         skeleton = pcv.morphology.skeletonize(mask=plant_mask)

         # The output of this step can look like it's almost totally black since the lines are
         # so thin and the image is large.
```

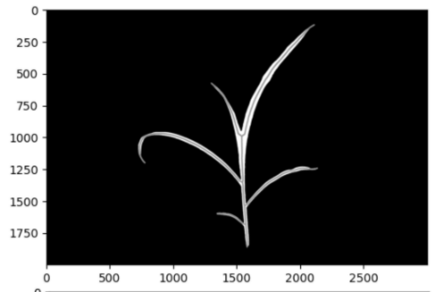


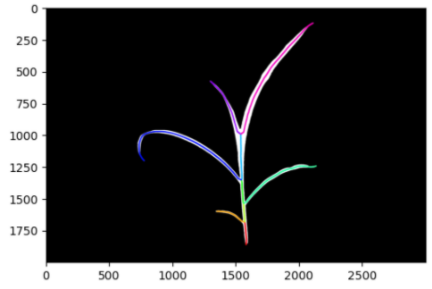
Block 16 refines the skeleton of the plant by allowing you to change line thickness and remove unnecessary spikes or fragments on the skeleton. When this block is finished running, you will see four images as the result of this block of code, showing how the plant skeleton structure is refined and then converted back to a cleaner image of the plant.

```
In [16]: # Adjust line thickness with the global line thickness parameter (default = 5),
         # and provide binary mask of the plant for debugging. NOTE: the objects and
         # hierarchies returned will be exactly the same but the debugging image (segmented_img)
         # will look different.
         pcv.params.line_thickness = 10

         # Prune the skeleton

         # Inputs:
         #   skel_img = Skeletonized image
         #   size     = Pieces of skeleton smaller than `size` should get removed. (Optional) Default `size=0`.
         #   mask     = Binary mask for debugging (optional). If provided, debug images will be overlaid on the mask.

         pruned, seg_img, edge_objects = pcv.morphology.prune(skel_img=skeleton, size=100, mask=plant_mask)
         #                                                                                ^
         #                                                                                |
         #                                                            Increase size to remove the extra, small spikes on
         #                                                            the skeletonized plant but don't increase to the
         #                                                            point where actual leaves get removed.
```



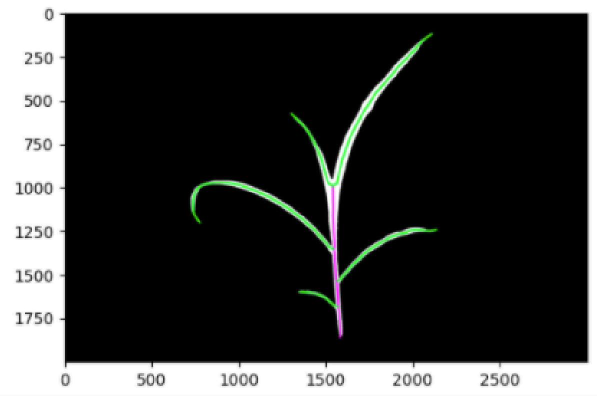






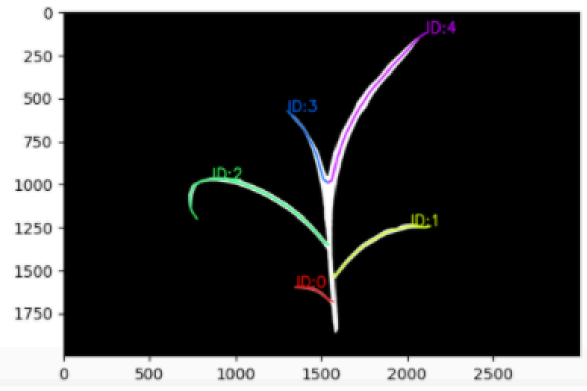
Block 17 differentiates between the stem (shown in pink below) and leaves (shown in green below) of the plant.

In [17]:
```
# Sort segments into leaf objects and stem objects

# Inputs:
#   skel_img  = Skeletonized image
#   objects   = List of contours
#   mask      = (Optional) binary mask for debugging. If provided, debug image
#                  will be overlaid on the mask.

leaf_obj, stem_obj = pcv.morphology.segment_sort(skel_img=pruned,
                                                 objects=edge_objects,
                                                 mask=plant_mask)
```



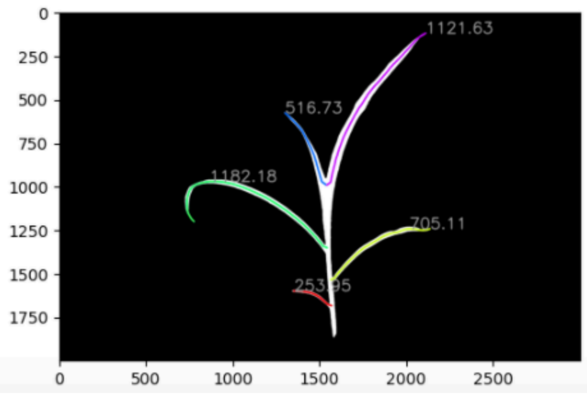
Block 18 labels each leaf with its own leaf ID.

In [18]:
```
# Similar to line thickness, there are optional text size and text thickness parameters
# that can be adjusted to better suit images or varying sizes.
pcv.params.text_size=3 # (default text_size=.55)
pcv.params.text_thickness=8 # (defaul text_thickness=2)

# Identify segments

# Inputs:
#   skel_img  = Skeletonized image
#   objects   = List of contours
#   mask      = (Optional) binary mask for debugging. If provided, debug image
#                  will be overlaid on the mask.

segmented_img, labeled_img = pcv.morphology.segment_id(skel_img=skeleton,
                                                       objects=leaf_obj,
                                                       mask=plant_mask)
```



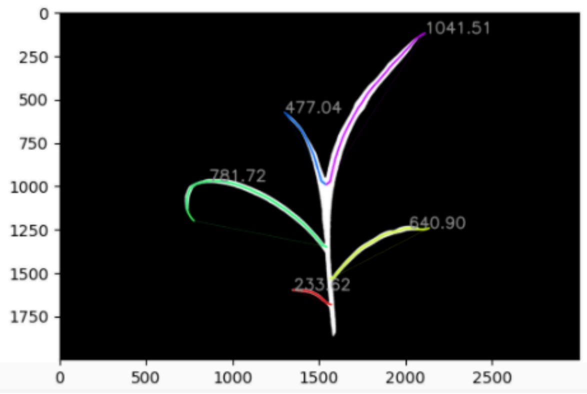
Block 19 measures path length of the leaves.

```
In [19]: # Measure path lengths of segments

         # Inputs:
         #   segmented_img = Segmented image to plot lengths on
         #   objects       = List of contours

         labeled_img  = pcv.morphology.segment_path_length(segmented_img=segmented_img,
                                                           objects=leaf_obj)
```



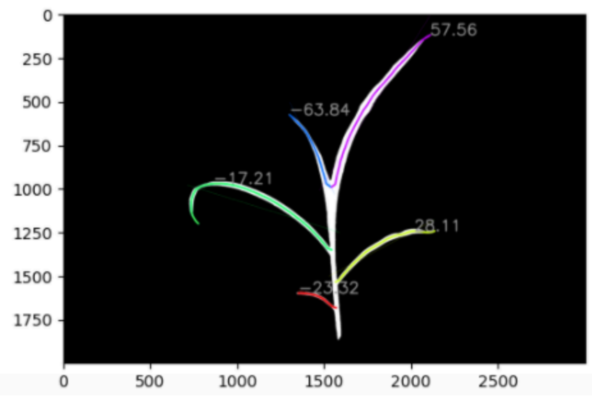
Block 20 measures the Euclidean distance of the leaves (necessary for the final two calculations).

```
In [20]: # Measure euclidean distance of segments

         # Inputs:
         #   segmented_img = Segmented image to plot lengths on
         #   objects       = List of contours

         labeled_img = pcv.morphology.segment_euclidean_length(segmented_img=segmented_img,
                                                               objects=leaf_obj)
```



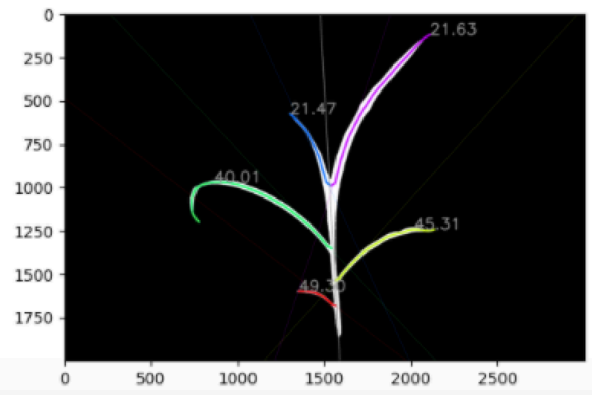
Block 21 measures the angle of segments.

```
In [21]: # Measure the angle of segments

         # Inputs:
         #   segmented_img = Segmented image to plot angles on
         #   objects       = List of contours

         labeled_img = pcv.morphology.segment_angle(segmented_img=segmented_img,
                                                    objects=leaf_obj)
```

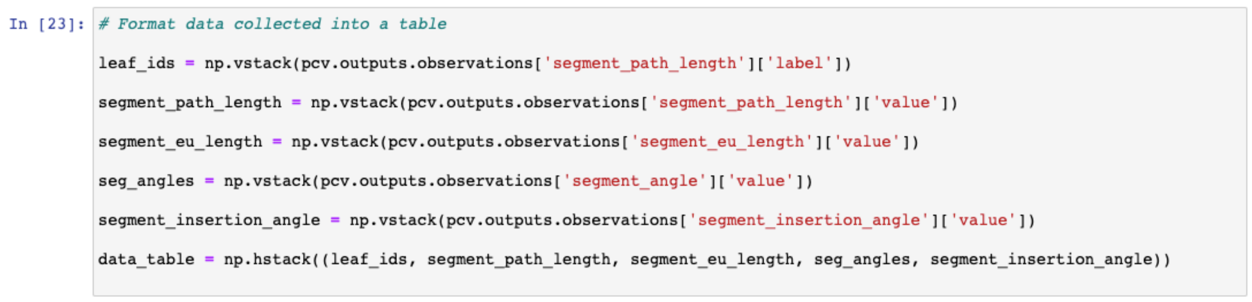


Block 22 measures the leaf insertion angle.

```
In [22]: # Measure the leaf insertion angles
         # NOTE: This function is slow and will likely take up to 2 minutes to run

         # Inputs:
         #   skel_img         = Skeletonize image
         #   segmented_img    = Segmented image to plot insertion angles on
         #   leaf_objects     = List of leaf contours
         #   stem_objects     = List of stem objects
         #   size             = Size of the inner portion of each leaf to find a linear regression line

         labeled_img = pcv.morphology.segment_insertion_angle(skel_img=skeleton,
                                                              segmented_img=segmented_img,
                                                              leaf_objects=leaf_obj,
                                                              stem_objects=stem_obj,
                                                              size=90)
```

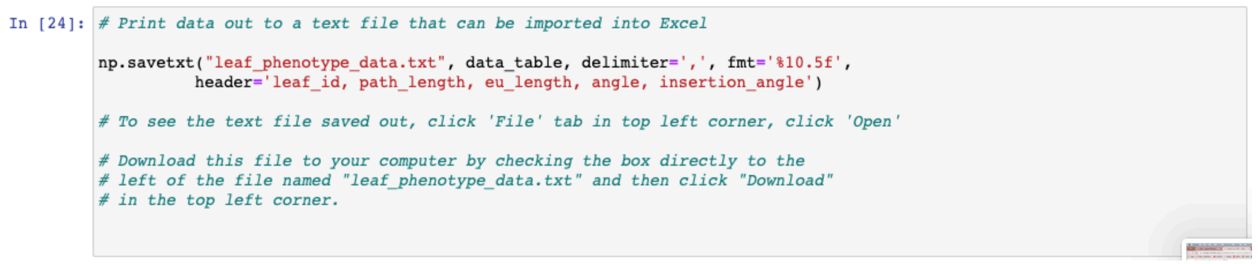


Block 23 inserts the data into a table.



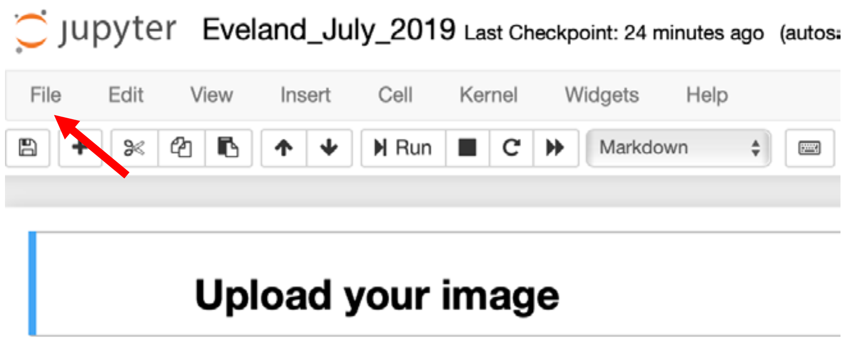
```
In [23]: # Format data collected into a table

         leaf_ids = np.vstack(pcv.outputs.observations['segment_path_length']['label'])

         segment_path_length = np.vstack(pcv.outputs.observations['segment_path_length']['value'])

         segment_eu_length = np.vstack(pcv.outputs.observations['segment_eu_length']['value'])

         seg_angles = np.vstack(pcv.outputs.observations['segment_angle']['value'])

         segment_insertion_angle = np.vstack(pcv.outputs.observations['segment_insertion_angle']['value'])

         data_table = np.hstack((leaf_ids, segment_path_length, segment_eu_length, seg_angles, segment_insertion_angle))
```

Block 24 exports the data table to a text file that can be read and saved to your own computer.



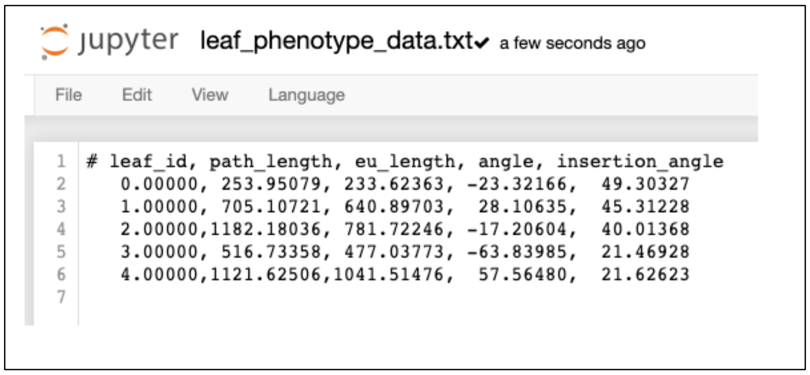
```
In [24]: # Print data out to a text file that can be imported into Excel

         np.savetxt("leaf_phenotype_data.txt", data_table, delimiter=',', fmt='%10.5f',
                   header='leaf_id, path_length, eu_length, angle, insertion_angle')

         # To see the text file saved out, click 'File' tab in top left corner, click 'Open'

         # Download this file to your computer by checking the box directly to the
         # left of the file named "leaf_phenotype_data.txt" and then click "Download"
         # in the top left corner.
```

7. After running the last block of code, you can download the data resulting from your analysis, by doing the following:
   7.1. In the top, left-hand corner of the activity, click **File** (indicated by the red arrow in the screenshot below).


jupyter Eveland_July_2019 Last Checkpoint: 24 minutes ago (autos
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
Upload your image

   7.2. Click **Open…** from the drop-down menu. A new tab should open in your browser.

7.3. Find the file **leaf_phenotype_data.txt**. Double click on this file. A new tab opens showing a Table with the data including the computational measurements of the sample image. The first column indicates the ID of each leaf. The parameter you are interested in, the leaf angle, is represented by the insertion angle (last column), which is calculated individually for every leaf on the plant. The image below shows you an example of a text file with data.



```
# leaf_id, path_length, eu_length, angle, insertion_angle
   0.00000, 253.95079, 233.62363, -23.32166,  49.30327
   1.00000, 705.10721, 640.89703,  28.10635,  45.31228
   2.00000,1182.18036, 781.72246, -17.20604,  40.01368
   3.00000, 516.73358, 477.03773, -63.83985,  21.46928
   4.00000,1121.62506,1041.51476,  57.56480,  21.62623
```

7.4. To download the data, click on **File** (upper left corner) and from the drop-down menu click **Download.** The downloaded file will show on your screen (bottom left). Open the text file by clicking on arrow (^) and **Open**. You will see the data again. Click on **File**, and then **Save As.** In the new tab choose a location to save the data from the sample image.

7.5. The text file that you downloaded can be imported into Excel. See the following section on How to analyze your own data for this step.

7.6. If required by your instructor, enter or copy/paste the data into the assigned data table.

Step 5. Analyze your own image

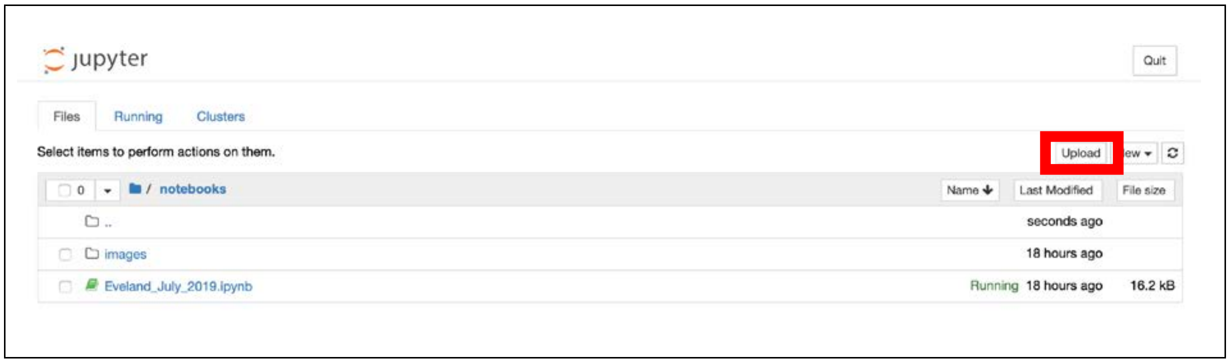
1. Access the photos you took of your seedlings either stored in your computer or phone.
2. If your photos are in your phone, move them to the computer that you will use to run PlantCV.
3. Upload the photos to PlantCV:

3.1. Back in the activity window (https://mybinder.org/v2/gh/danforthcenter/Eveland_NSF_Outreach/master?filepath=index.ipynb), click **File** in the top, left corner.

3.2. Click **Open…** from the drop-down menu. A new tab should open in your browser.

3.3. In the new tab, click the folder named **images**.

3.4. In the top, right-hand corner click **Upload**, indicated with a red box below.

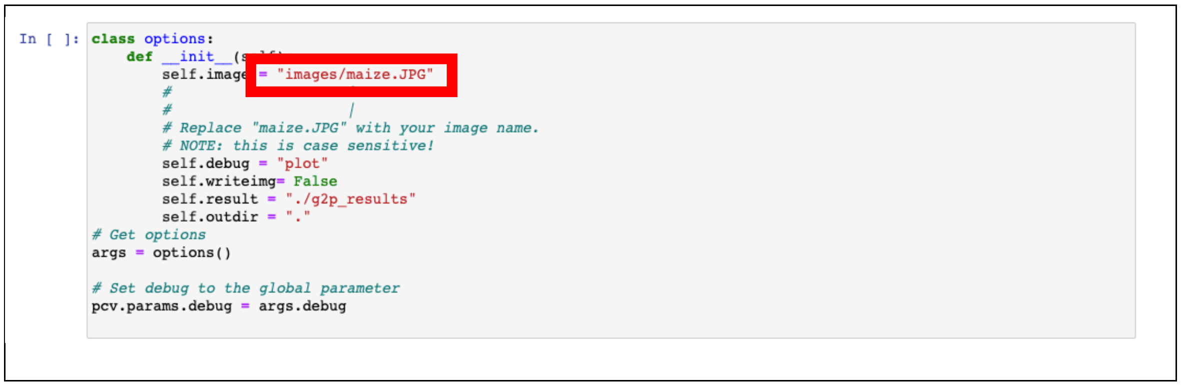


Select items to perform actions on them.

0 / notebooks			
..		seconds ago	
images		18 hours ago	
Eveland_July_2019.ipynb	Running	18 hours ago	16.2 kB

3.5. This will open a new window where you can browse and select the images you took.
3.6. Your image files will appear in the **images** folder. To complete the upload, click the **Upload** button to the right of each file name. Then close the new browser tab and return to the main activity page.

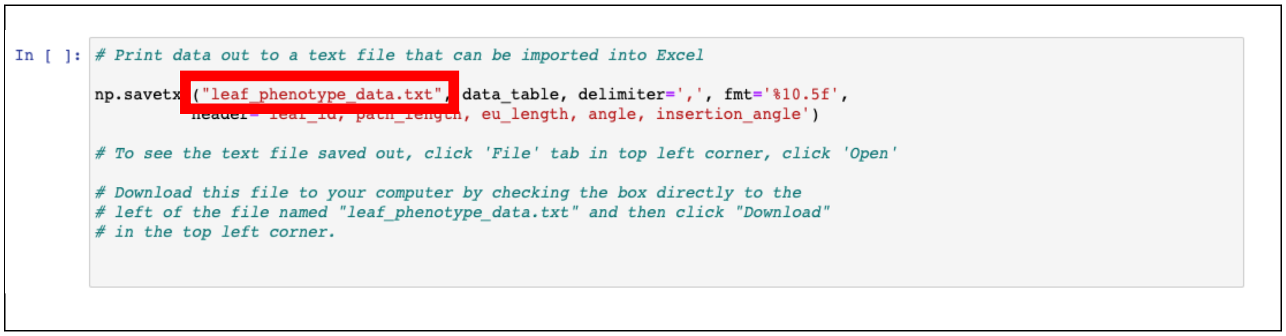
4. Analyze the images one at a time:
   4.1. Repeat the steps of image analysis the same way you did with the sample image starting in page 4 of this protocol.
   4.2. Keep in mind that you need to make some changes to the code. Take care to read the green, italicized comments about how to change the analysis parameters for your image within some blocks. Many of the blocks of code, however, will not need updating.
   4.3. The first change is in Block 2, where you have to **change the file name** in the code (highlighted with a red box below) to match the file name of the image you want to analyze (e.g. 138_4_SA.jpg). Be sure to keep the quotation marks.



```
In [ ]: class options:
            def __init__(self):
                self.image = "images/maize.JPG"
                #
                #                    |
                # Replace "maize.JPG" with your image name.
                # NOTE: this is case sensitive!
                self.debug = "plot"
                self.writeimg= False
                self.result = "./g2p_results"
                self.outdir = "."
        # Get options
        args = options()

        # Set debug to the global parameter
        pcv.params.debug = args.debug
```

   4.4. Continue running the blocks. Adjust the Region of Interest (ROI) as necessary (Block 13, page 9).
   4.5. IMPORTANT, before running the final block of text, Block 24 in this protocol (page 15), you need to change the name of your data file to one corresponding to your image. In this last block of code replace the red text reading **leaf_phenotype_data.txt** (highlighted with a red box below) with your own file name: **(genotype number_seedling number_first letters of first and last names)** (ex. (e.g. 138_4_SA.txt).

The new file name will look similar to the original image file name; however, be sure to use the **.txt** extension for the new file name. Keep the quotation marks around the file name as well.



```
In [ ]: # Print data out to a text file that can be imported into Excel

        np.savetxt("leaf_phenotype_data.txt", data_table, delimiter=',', fmt='%10.5f',
                   header='leaf_id, path_length, eu_length, angle, insertion_angle')

        # To see the text file saved out, click 'File' tab in top left corner, click 'Open'

        # Download this file to your computer by checking the box directly to the
        # left of the file named "leaf_phenotype_data.txt" and then click "Download"
        # in the top left corner.
```

5. After the last block of code has been run, download your data to your computer repeating the steps for the sample image in page 15 sections 6.
   5.1. Find the data file that was generated (e.g. 138_4_SA.txt).
   5.2. Download this text file to your computer.
   5.3. You can save this file as an Excel file.
   5.4. Upload the data files from each of the images you analyzed to your Google Classroom space.
6. Answer the questions comparing the manual and image-based measurements of leaf angle among genotypes and within genotypes in the document provided in your Google Classroom.

RESOURCES

Documents on how to use PlantCV for a variety of image analyses are available at:

https://plantcv.readthedocs.io/en/stable/

If you have any questions contact:
Sandra Arango-Caro
Sarango-caro@danforthcenter.org