triangle_graphing_21_may_2024_abridged

[Note that this Portable Format Document (to print out onto pieces of white paper which are each 8.5 inches wide and 11 inches tall using black ink, sans-serif font, and 11 point font size) contains plain-text content only and that not all the content which is featured on the web page named Karlina Object dot WordPress dot Com forward slash Triangle Graphing is featured also in this document].

https://karlinaobject.wordpress.com/triangle_graphing/

The final draft version of this document was published on 21_MAY_2024.

TRIANGLE_GRAPHING

The single web page application featured in this tutorial web page allows the user to select four integer input values to use as the coordinates for three unique points which comprise a non-degenerate triangle which will be drawn inside of an HTML5 canvas element whose dimensions are 750 pixels on each side and which is scaled to depict a Cartesian grid whose origin is the center of the canvas and whose maximum x-axis and y-axis values are 100.

Note that the software application featured in this tutorial web page is an older version of the software application which is featured on the web page whose Uniform Resource Locator is as follows: https://karbytesforlifeblog.wordpress.com/triangle_graphing_two/

To view hidden text inside of the preformatted text boxes below, scroll horizontally.

TRIANGLE_GRAPHING Software Application Components

Hyper-Text-Markup-Language_file: https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/triangle_graphing.html

Cascading-Style-Sheet_file: https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/karbytes_aesthetic.css

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_0/main/triangle_graphing.js

---

TRIANGLE_GRAPHING Interface (after selecting inputs)

---

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_0/main/triangle_graphing_web_page_interface_after_selecting_input_values.png

---

TRIANGLE_GRAPHING Interface (after clicking GENERATE)

---

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_0/main/triangle_graphing_web_page_interface_after_clicking_generate_button.png

---

TRIANGLE_GRAPHING Interface (after clicking RESET)

---

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_0/main/triangle_graphing_web_page_interface_after_clicking_reset_button.png

---

TRIANGLE_GRAPHING Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the TRIANGLE_GRAPHING web page application. Copy the CSS code from the preformatted text box below or from the source code file which is linked below into a text editor and save that file as karbytes_aesthetic.css.

Cascading-Style-Sheet_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/karbytes_aesthetic.css

---

```
/**
 * file: karbytes_aesthetic.css
 * type: Cascading-Style-Sheet
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/** Make the page background BLACK, the text orange and monospace, and the page content width 800 pixels or less. */
body {
        background: #000000;
        color: #ff9000;
        font-family: monospace;
        font-size: 16px;
        padding: 10px;
        width: 800px;
}

/** Make input elements and select elements have an orange rounded border, a BLACK background, and orange monospace text. */
input, select {
        background: #000000;
        color: #ff9000;
        border-color: #ff9000;
        border-width: 1px;
        border-style: solid;
        border-radius: 5px;
        padding: 10px;
        appearance: none;
        font-family: monospace;
        font-size: 16px;
}

/** Invert the text color and background color of INPUT and SELECT elements when the cursor (i.e. mouse) hovers over them. */
input:hover, select:hover {
        background: #ff9000;
        color: #000000;
}

/** Make table data borders one pixel thick and CYAN. Give table data content 10 pixels in
padding on all four sides. */
td {
        color: #00ffff;
        border-color: #00ffff;
        border-width: 1px;
        border-style: solid;
        padding: 10px;
}

/** Set the text color of elements whose identifier (id) is "output" to CYAN. */
#output {
        color: #00ffff;
}

/** Set the text color of elements whose class is "console" to GREEN and make the text
background of those elements BLACK. */
.console {
        color: #00ff00;
        background: #000000;
}
```

TRIANGLE_GRAPHING Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the TRIANGLE_GRAPHING web page application. Copy the HTML code from the source code file which is linked below into a text editor and save that file as triangle_graphing.html. Use a web browser such as Firefox to open that HTML file (and ensure that the JavaScript and Cascading-Style-Sheet files are in the same file directory as the HTML file).

Note that the contents of the HTML file are not displayed in a preformatted text box on this web page due to the fact that the WordPress server makes no distinction between HTML code which is encapsulated inside of a preformatted text box and WordPress web page source code.

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/triangle_graphing.html

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/triangle_graphing_html_code_screenshot.png

---

TRIANGLE_GRAPHING JavaScript Code

The following JavaScript (JS) code defines the functions which control the behavior of the TRIANGLE_GRAPHING web page application. Copy the JS code from the preformatted text box below or from the source code file which is linked below into a text editor and save that file as triangle_graphing.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_0/main/triangle_graphing.js

---

```
/**
 * file: triangle_graphing.js
 * type: JavaScript
 * author: karbytes
 * date: 21_JULY_2023
 * license: PUBLIC_DOMAIN
 */

/**
 * Return a String type value which describes the number of milliseconds which have elapsed since the Unix Epoch.
 *
 * Note that the Unix Epoch is 01_JANUARY_1970 at 0 hours, 0 minutes, 0 seconds, and 0 seconds
 * (i.e. 00:00:00) (i.e. midnight) (Coordinated Universal Time (UTC)).
 *
 * @return {String} text which denotes the number of milliseconds which have elapsed since the Unix Epoch.
 */
function generate_time_stamp() {
    const milliseconds_elapsed_since_unix_epoch = Date.now();
    return milliseconds_elapsed_since_unix_epoch + " milliseconds since midnight on 01_JANUARY_1970.";
}
```

```
/**
 * Return a String type value which is used to instantiate a paragraph type web page element
such that
 * the String type value which is passed into this function as its only input is that paragraph
element's
 * inner HTML content.
 *
 * Note that the String type constant variable values are broken up into single-character String
type values
 * to avoid causing the WordPress web page editor to interpret HTML tags in the web page body
with
 * source code which is hosted on that web page inside of PRE (preformatted) web page
elements.
 *
 * @param {String} inner_HTML is assumed to be plain text or HTML content.
 *
 * @return {String} a sequence of text characters which is used to instantiate a paragraph (P)
web page element.
 */
function generate_paragraph_html_element(inner_html) {
    const opening_paragraph_tag = ("), closing_paragraph_tag = (");
    try {
        if (typeof inner_html.length !== "number") throw 'The expression (typeof inner_html.length
!== "number") was evaluated to be true.';
        return opening_paragraph_tag + inner_html + closing_paragraph_tag;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
generate_paragraph_html_element(inner_html): " + exception);
    }
}

/**
 * Return a String type value which is used to instantiate a select type web page element such
that
 * the String type value which is passed into this function as its only input is that select menu
element's
 * id property value.
 *
 * When clicked on, the select menu interface element will expand to display a list of all integers
which are
 * no smaller than -100 and no larger than 100 in ascending order (with the smallest integer
option at the top
```

```
 * of the list).
 *
 * @param {String} select_id is assumed to be either
 *                "a_x_menu" or else
 *                "a_y_menu" or else
 *                "b_x_menu" or else
 *                "b_y_menu" or else
 *                "c_x_menu" or else
 *                "c_y_menu".
 *
 * @return {String} a sequence of text characters which is used to instantiate an expandable list
menu (SELECT) web page element.
 */
function generate_coordinate_menu_select_html_element(select_id) {
   let select_menu = '', option = '', i = 0;
   try {
      if (typeof select_id.length !== "number") throw 'The expression (typeof select_id.length !==
"number") was evaluated to be true.';
      if ((select_id !== "a_x_menu") && (select_id !== "a_y_menu") &&
         (select_id !== "b_x_menu") && (select_id !== "b_y_menu") &&
         (select_id !== "c_x_menu") && (select_id !== "c_y_menu"))
      throw 'select_id must either be "a_x_menu" or else "a_y_menu" or ' +
           'else "b_x_menu" or else "b_y_menu" or ' +
           'else "c_x_menu" or else "c_y_menu".';
      select_menu = ('');
      for (i = -100; i <= 100; i += 1) {
         if (i === 0) option = ('');
         else option = ('');
         option += (i + (''));
         select_menu += option;
      }
      select_menu += ('');
      return select_menu;

   }
   catch(exception) {
      console.log("An exception to normal functioning occurred during the runtime of
generate_coordinate_menu_select_html_element(select_id): " + exception);
   }
}

/**
 * Return the String type value of the selected menu option of a SELECT menu element.
 *
 * Assume that select_menu_identifier is a String type value and the id of an existing select
HTML element.
 *
 * @param {String} select_menu_identififier is assumed to be the id of an existing SELECT
menu web page element.
 *
 * @return {String} value of an OPTION of the SELECT whose id is select_menu_identifier.
 */
function get_selected_menu_option_value(select_menu_identifier) {
    try {
        let menu_object = {}, options_array = [], selected_option_index = 0, selected_option_object
= {}, selected_option_value;
        if (arguments.length !== 1) throw "Error: exactly one function input is required.";
        if (typeof arguments[0] !== "string") throw "Error: select_menu_identifier is required to be a
String type data value.";
        menu_object = document.getElementById(select_menu_identifier);
        options_array = menu_object.options;
        selected_option_index = menu_object.selectedIndex;
        selected_option_object = options_array[selected_option_index];
        selected_option_value = selected_option_object.value
        return selected_option_value;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
get_selected_menu_option(select_menu_identifier): " + exception);
    }
}

/**
 * Draw a line segment whose thickness is one pixel and whose color is black from the middle
point of the left edge
 * of the HTML canvas whose id is "cartesian_plane" to the middle of the right edge of that
canvas.
 *
 * Assume that the length of each one of the four sides of that canvas is 750 pixels.
 */
function draw_horizontal_line_through_middle_of_canvas() {
    const CANVAS_SIDE_LENGTH = 750;
    let canvas = undefined, context = undefined, canvas_midpoint = 0;
    try {
        canvas = document.getElementById("cartesian_plane");
        if (canvas.width !== canvas.height) throw "The expression (canvas.width !==
canvas.height) was evaluated to be true.";
```

```
        if (canvas.width !== CANVAS_SIDE_LENGTH) throw "The expression (canvas.width !== CANVAS_SIDE_LENGTH) was evaluated to be true.";
        canvas_midpoint = (canvas.width / 2);
        canvas_midpoint = parseInt(canvas_midpoint);
        context = canvas.getContext("2d");
        context.strokeStyle = "#000000";
        context.lineWidth = 1;
        context.beginPath();
        context.moveTo(0, canvas_midpoint); // the middle point of the left edge of the square canvas
        context.lineTo((canvas_midpoint * 2), canvas_midpoint); // the middle point of the right edge of the square canvas
        context.stroke();
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in draw_horizontal_line_through_middle_of_canvas(): " + exception);
    }
}

/**
 * Draw a line segment whose thickness is one pixel and whose color is black from the middle point of the top edge
 * of the HTML canvas whose id is "cartesian_plane" to the middle of the bottom edge of that canvas.
 *
 * Assume that the length of each one of the four sides of that canvas is 750 pixels.
 */
function draw_vertical_line_through_middle_of_canvas() {
    const CANVAS_SIDE_LENGTH = 750;
    let canvas = undefined, context = undefined, canvas_midpoint = 0;
    try {
        canvas = document.getElementById("cartesian_plane");
        if (canvas.width !== canvas.height) throw "The expression (canvas.width !== canvas.height) was evaluated to be true.";
        if (canvas.width !== CANVAS_SIDE_LENGTH) throw "The expression (canvas.width !== CANVAS_SIDE_LENGTH) was evaluated to be true.";
        canvas_midpoint = (canvas.width / 2);
        canvas_midpoint = parseInt(canvas_midpoint);
        context = canvas.getContext("2d");
        context.strokeStyle = "#000000";
        context.lineWidth = 1;
        context.beginPath();
        context.moveTo(canvas_midpoint, 0); // the middle point of the top edge of the square
canvas
        context.lineTo(canvas_midpoint, (canvas_midpoint * 2)); // the middle point of the bottom
edge of the square canvas
        context.stroke();
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in
draw_vertical_line_through_middle_of_canvas(): " + exception);
    }
}

/**
 * Respond to the event of the RESET button being clicked or the web page being loaded by a
web browser.
 */
function initialize_application() {
    let cartesian_plane_canvas = "";
    let time_stamped_message = "", initial_output_message = "";
    let canvas_container_div = undefined, output_div = undefined, events_log_div = undefined,
generate_button_container_paragraph = undefined;
    let a_x_menu_container_paragraph = undefined, a_y_menu_container_paragraph =
undefined;
    let b_x_menu_container_paragraph = undefined, b_y_menu_container_paragraph =
undefined;
    let c_x_menu_container_paragraph = undefined, c_y_menu_container_paragraph =
undefined;
    let generate_button_container = "";
    try {
        // Populate the "event_log" div with a time stamped message indicating that this function
was called.
        time_stamped_message = ("The function named initialize_application() was called at time:
" + generate_time_stamp());
        console.log(time_stamped_message);
        time_stamped_message = generate_paragraph_html_element(time_stamped_message);
        events_log_div = document.getElementById("events_log");
        events_log_div.innerHTML = time_stamped_message;
        // Populate the "output" div with placeholder text.
        output_div = document.getElementById("output");
        output_div.innerHTML = generate_paragraph_html_element("This sentence will disappear
as a result of the GENERATE button being clicked.");
        // Populate the "canvas_container" div with a canvas web page element.
        cartesian_plane_canvas = ((") + ("));
        canvas_container_div = document.getElementById("canvas_container");
        canvas_container_div.innerHTML =
generate_paragraph_html_element(cartesian_plane_canvas);
        // Draw the horizontal axis of a Cartesian plane through the center of the square canvas.
        draw_horizontal_line_through_middle_of_canvas();
        // Draw the vertical axis of a Cartesian plane through the center of the square canvas.
        draw_vertical_line_through_middle_of_canvas();
        // Populate the "a_x_menu_container" paragraph element with a select menu for choosing
an integer value for the X property of POINT object A.
        a_x_menu_container_paragraph = document.getElementById("a_x_menu_container");
        a_x_menu_container_paragraph.innerHTML = ('A.X := ' +
generate_coordinate_menu_select_html_element("a_x_menu") + '. // horizontal position of
two-dimensional POINT labeled A.');
        // Populate the "a_y_menu_container" paragraph element with a select menu for choosing
an integer value for the Y property of POINT object A.
        a_y_menu_container_paragraph = document.getElementById("a_y_menu_container");
        a_y_menu_container_paragraph.innerHTML = ('A.Y := ' +
generate_coordinate_menu_select_html_element("a_y_menu") + '. // vertical position of
two-dimensional POINT labeled A.');
        // Populate the "b_x_menu_container" paragraph element with a select menu for choosing
an integer value for the X property of POINT object B.
        b_x_menu_container_paragraph = document.getElementById("b_x_menu_container");
        b_x_menu_container_paragraph.innerHTML = ('B.X := ' +
generate_coordinate_menu_select_html_element("b_x_menu") + '. // horizontal position of
two-dimensional POINT labeled B.');
        // Populate the "B_y_menu_container" paragraph element with a select menu for choosing
an integer value for the Y property of POINT object B.
        b_y_menu_container_paragraph = document.getElementById("b_y_menu_container");
        b_y_menu_container_paragraph.innerHTML = ('B.Y := ' +
generate_coordinate_menu_select_html_element("b_y_menu") + '. // vertical position of
two-dimensional POINT labeled B.');
        // Populate the "c_x_menu_container" paragraph element with a select menu for choosing
an integer value for the X property of POINT object C.
        c_x_menu_container_paragraph = document.getElementById("c_x_menu_container");
        c_x_menu_container_paragraph.innerHTML = ('C.X := ' +
generate_coordinate_menu_select_html_element("c_x_menu") + '. // horizontal position of
two-dimensional POINT labeled C.');
        // Populate the "C_y_menu_container" paragraph element with a select menu for choosing
an integer value for the Y property of POINT object B.
        c_y_menu_container_paragraph = document.getElementById("c_y_menu_container");
        c_y_menu_container_paragraph.innerHTML = ('C.Y := ' +
generate_coordinate_menu_select_html_element("c_y_menu") + '. // vertical position of
two-dimensional POINT labeled C.');
        // Populate the "generate_button_container" paragraph element with a button input web
page element which calls the function named generate_triangle_using_input_coordinates().
```

```
        generate_button_container = document.getElementById("generate_button_container");
        generate_button_container.innerHTML = (");
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
initialize_application(): " + exception);
    }
}

/**
 * Compute the approximate square root of input such that the output has an arbitrary number of
significant digits.
 * The product, approximate_square_root(input) * approximate_square_root(input), is
approximately equal to input.
 *
 * @param {Number} input is assumed to be a nonnegative integer.
 *
 * @return {Number} the approximate square root of input.
 */
function approximate_square_root(input) {
    let n = 0, a = 0, b = 0, c = 0;
    try {
        if (arguments.length !== 1) throw "exactly one function argument is required.";
        if (typeof arguments[0] !== "number") throw "the function argument must be a Number type
value.";
        if (input  c) {
            a = (a + b) / 2;
            b = n / a;
        }
        return a;
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in
approximate_square_root(input): " + exception);
        return 0;
    }
}

/**
 * Use the Distance Formula to calculate the nonnegative real number distance between planar
points A and B.
 *
 * distance_formula(A, B) = square_root( ((A.x - B.x) ^ 2) + ((A.y - B.y) ^ 2) )
 *
 * @param {Object} A is assumed to be an Object type data value with the following properties:
 *       {Number} X is assumed to be an integer no smaller than -100 and no larger than 100.
 *       {Number} Y is assumed to be an integer no smaller than -100 and no larger than 100.
 *
 * @param {Object} B is assumed to be an Object type data value with the following properties:
 *       {Number} X is assumed to be an integer no smaller than -100 and no larger than 100.
 *       {Number} Y is assumed to be an integer no smaller than -100 and no larger than 100.
 *
 * @return {Number} the length of the shortest path between planar points A and B.
 */
function compute_distance_between_two_planar_points(A, B) {
   let horizontal_difference = 0, vertical_difference = 0;
   try {
      if (arguments.length !== 2) throw "exactly two function arguments are required.";
      if (!is_point(A)) throw "A must be an object whose data properties are as follows: { X :
integer in range [-100,100], Y : integer in range [-100,100] }.";
      if (!is_point(B)) throw "B must be an object whose data properties are as follows: { X :
integer in range [-100,100], Y : integer in range [-100,100] }.";
      horizontal_difference = A.X - B.X
      vertical_difference = A.Y - B.Y;
      return approximate_square_root((horizontal_difference * horizontal_difference) +
(vertical_difference * vertical_difference));
   }
   catch(exception) {
      console.log("An exception to expected functioning occurred in
compute_distance_between_two_planar_points(A, B): " + exception);
      return 0;
   }
}

/**
 * Compute the rate at which the y-value changes in relation to the x-value in the function whose
graph
 * is the line which completely overlaps the line segment whose endpoints are A and B.
 *
 * // y := f(x),
 * // b := f(0),
 * // f is a function whose input is an x-axis position and whose output is a y-axis position.
 * y := mx + b.
 *
 * // m is a constant which represents the rate at which y changes in relation to x changing.
 * m := (y - b) / x.
 *
 * // m represents the difference of the two y-values divided by the difference of the two x-values.
 * m := (A.Y - B.Y) / (A.X - B.X).
 *
 * @param {Object} A is assumed to be an Object type data value with the following properties:
 *       {Number} X is assumed to be an integer no smaller than -100 and no larger than 100.
 *       {Number} Y is assumed to be an integer no smaller than -100 and no larger than 100.
 *
 * @param {Object} B is assumed to be an Object type data value with the following properties:
 *       {Number} X is assumed to be an integer no smaller than -100 and no larger than 100.
 *       {Number} Y is assumed to be an integer no smaller than -100 and no larger than 100.
 *
 * @return {Number} the slope of the shortest path between planar points A and B.
 */
function get_slope_of_line_segment(A, B) {
    let vertical_difference = 0, horizontal_difference = 0;
    try {
        if (arguments.length !== 2) throw "exactly two function arguments are required.";
        if (!is_point(A)) throw "A must be an object whose data properties are as follows: { X :
integer in range [-100,100], Y : integer in range [-100,100] }.";
        if (!is_point(B)) throw "B must be an object whose data properties are as follows: { X :
integer in range [-100,100], Y : integer in range [-100,100] }.";
        horizontal_difference = A.X - B.X;
        vertical_difference = A.Y - B.Y;
        console.log("horizontal_difference := " + horizontal_difference + '.');
        console.log("vertical_difference := " + vertical_difference + '.');
        return (vertical_difference / horizontal_difference);
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in
get_slope_of_line_segment(A, B): " + exception);
        return 0;
    }
}

/**
 * Determine whether or not a given input value is a valid planar point object (as defined in the
generate_triangle_using_input_coordinates() function).
 *
 * @param {Object} input is assumed to be an Object type data value with the following
properties:
 *       {Number} X is assumed to be an integer no smaller than -100 and no larger than 100.
 *       {Number} Y is assumed to be an integer no smaller than -100 and no larger than 100.
 *
 * @return {Boolean} true if input satisfies the conditions defined above; false otherwise.
 */
```

```
function is_point(input) {
    // let properties = undefined, count = 0;
    try {
        if (arguments.length !== 1) throw "exactly one function argument (labeled input) is
required.";
        if (typeof input !== "object") throw "input must be an Object type value.";
        if (typeof input.X !== "number") throw "the X property of input must be a Number type
value.";
        if (typeof input.Y !== "number") throw "the Y property of input must be a Number type
value.";
        if (Math.floor(input.X) !== input.X) throw "the X property of the input object must be a whole
number value.";
        if (Math.floor(input.Y) !== input.Y) throw "the Y property of the input object must be a whole
number value.";
        if ((input.X  100)) throw "the X of the input object must be no smaller than -100 and no
larger than 100.";
        if ((input.Y  100)) throw "the X of the input object must be no smaller than -100 and no
larger than 100.";
        /*
        // This is commented out due to the fact that karbytes wanted to allow indefinitely many
function properties to be added to POINT "type" objects.
        for (let properties in input) count += 1;
        if (count !== 2) throw "input must be an object consisting of exactly two properties.";
        */
        return true;
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in is_point(input): " +
exception);
        return false;
    }
}

/**
 * Generate an Object type data value which represents a position on a two-dimensional
Cartesian grid.
 *
 * If an exception is thrown while the try-catch block is being executed, return a POINT whose
coordinate values are both zero.
 *
 * @param {Number} X is assumed to be an integer no smaller than -100 and no larger than
100.
 *
 * @param {Number} Y is assumed to be an integer no smaller than -100 and no larger than
100.
 *
 * @return {Object} an object consisting of exactly two key-value pairs named X and Y and
 *                exactly three functions named distance, slope, and description.
 */
function POINT(X,Y) {
    let distance = function(P) { return compute_distance_between_two_planar_points(this, P) };
    let slope = function(P) { return get_slope_of_line_segment(this, P) };
    let description = function() {
        return ('POINT(' + this.X + ',' + this.Y + ')');
    };
    try {
        if (arguments.length !== 2) throw "exactly two function arguments are required.";
        if (is_point({X:X,Y:Y})) return {X:X, Y:Y, DISTANCE:distance, SLOPE:slope,
DESCRIPTION:description};
        else throw "The expression (is_point({X:X,Y:Y})) was evaluated to be false.";
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in POINT(X,Y): " + exception);
        return {X:0, Y:0, DISTANCE:distance, SLOPE:slope, DESCRIPTION:description};
    }
}

/**
 * Generate an Object type data value which represents the triangular region of space whose
corners
 * are points represented by input POINT instances A, B, and C.
 *
 * If an exception is thrown while the try-catch block is being executed, return a POINT whose
coordinate values are both zero.
 *
 * A, B, and C are assumed to represent unique points in two-dimensional space.
 *
 * The area of the two-dimensional space whose boundaries are the shortest paths between
points A, B, and C
 * is assumed to be a positive real number quantity. (Therefore, A, B, and C are required to not
be located on the same line).
 *
 * @param {Object} A is assumed to be a value returned by the function POINT(X,Y).
 *
 * @param {Object} B is assumed to be a value returned by the function POINT(X,Y).
 *
 * @param {Object} C is assumed to be a value returned by the function POINT(X,Y).
 *
 * @return {Object} an object consisting of exactly three data attributes named A, B, and C
 *                and exactly nine function attributes named perimeter, area, angle_a, angle_b,
angle,
 *                length_ab, length_bc, length_ca, and description.
 */
function TRIANGLE(A,B,C) {
    let _A = {}, _B = {}, _C = {};
    let perimeter = function() { return (this.LENGTH_AB() + this.LENGTH_BC() +
this.LENGTH_CA()); };
    let area = function() {
        let s = 0.0, a = 0.0, b = 0.0, c = 0.0;
        s = this.PERIMETER() / 2; // s is technically referred to as the semiperimter of the triangle
which the caller TRIANGLE object of this function represents.
        a = this.LENGTH_BC(); // a represents the length of the line segment whose endpoints are
this.B and this.C.
        b = this.LENGTH_CA(); // b represents the length of the line segment whose endpoints are
this.C and this.A.
        c = this.LENGTH_AB(); // c represents the length of the line segment whose endpoints are
this.A and this.B.
        return Math.sqrt(s * (s - a) * (s - b) * (s - c)); // Use Heron's Formula to compute the area of
the triangle whose points are A, B, and C (and which are points of the caller TRIANGLE object
of this function represents).
    };
    let angle_a = function() {
        let a = 0.0, b = 0.0, c = 0.0, angle_opposite_of_a = 0.0, angle_opposite_of_b = 0.0,
angle_opposite_of_c = 0.0;
        a = this.LENGTH_BC(); // a represents the length of the line segment whose endpoints are
this.B and this.C.
        b = this.LENGTH_CA(); // b represents the length of the line segment whose endpoints are
this.C and this.A.
        c = this.LENGTH_AB(); // c represents the length of the line segment whose endpoints are
this.A and this.B.
        angle_opposite_of_a = Math.acos(((b * b) + (c * c) - (a * a)) / (2 * b * c)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        angle_opposite_of_b = Math.acos(((a * a) + (c * c) - (b * b)) / (2 * a * c)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        angle_opposite_of_c = Math.acos(((a * a) + (b * b) - (c * c)) / (2 * a * b)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        return angle_opposite_of_a; // in degrees (instead of in radians)
    };
    let angle_b = function() {
        let a = 0.0, b = 0.0, c = 0.0, angle_opposite_of_a = 0.0, angle_opposite_of_b = 0.0,
angle_opposite_of_c = 0.0;
```

```
        a = this.LENGTH_BC(); // a represents the length of the line segment whose endpoints are
this.B and this.C.
        b = this.LENGTH_CA() // b represents the length of the line segment whose endpoints are
this.C and this.A.
        c = this.LENGTH_AB(); // c represents the length of the line segment whose endpoints are
this.A and this.B.
        angle_opposite_of_a = Math.acos(((b * b) + (c * c) - (a * a)) / (2 * b * c)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        angle_opposite_of_b = Math.acos(((a * a) + (c * c) - (b * b)) / (2 * a * c)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        angle_opposite_of_c = Math.acos(((a * a) + (b * b) - (c * c)) / (2 * a * b)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        return angle_opposite_of_b; // in degrees (instead of in radians)
    };
    let angle_c = function() {
        let a = 0.0, b = 0.0, c = 0.0, angle_opposite_of_a = 0.0, angle_opposite_of_b = 0.0,
angle_opposite_of_c = 0.0;
        a = this.LENGTH_BC(); // a represents the length of the line segment whose endpoints are
this.B and this.C.
        b = this.LENGTH_CA() // b represents the length of the line segment whose endpoints are
this.C and this.A.
        c = this.LENGTH_AB(); // c represents the length of the line segment whose endpoints are
this.A and this.B.
        angle_opposite_of_a = Math.acos(((b * b) + (c * c) - (a * a)) / (2 * b * c)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        angle_opposite_of_b = Math.acos(((a * a) + (c * c) - (b * b)) / (2 * a * c)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        angle_opposite_of_c = Math.acos(((a * a) + (b * b) - (c * c)) / (2 * a * b)) * (180 / Math.PI); //
acos implies inverse of cosine (and Math.acos() returns a nonnegative number of radians)
        return angle_opposite_of_c; // in degrees (instead of in radians)
    };
    let length_ab = function(A,B) { return this.A.DISTANCE(this.B); };
    let length_bc = function(B,C) { return this.B.DISTANCE(this.C); };
    let length_ca = function(C,A) { return this.C.DISTANCE(this.A); };
    let description = function() {
        return ('TRIANGLE(A := ' + this.A.DESCRIPTION() + ', B:= ' + this.B.DESCRIPTION() + ', C
:= ' + this.C.DESCRIPTION() + ')');
    };
    try {
        if (arguments.length !== 3) throw "exactly three function arguments are required.";
        _A = POINT(A.X,A.Y);
        _B = POINT(B.X,B.Y);
        _C = POINT(C.X,C.Y);
        if ((_A.X === _B.X) && (_A.Y === _B.Y)) throw "A and B appear to represent the same
planar coordinates.";
        if ((_A.X === _C.X) && (_A.Y === _C.Y)) throw "A and C appear to represent the same
planar coordinates.";
        if ((_C.X === _B.X) && (_C.Y === _B.Y)) throw "C and B appear to represent the same
planar coordinates.";
        return {A:_A, B:_B, C:_C, PERIMETER:perimeter, AREA:area, ANGLE_A:angle_a,
ANGLE_B:angle_b, ANGLE_C:angle_c, LENGTH_AB:length_ab, LENGTH_BC:length_bc,
LENGTH_CA:length_ca, DESCRIPTION:description};
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in TRIANGLE(A,B,C): " +
exception);
        return {A:POINT(0,0), B:POINT(1,1), C:POINT(0,1), PERIMETER:perimeter, AREA:area,
ANGLE_A:angle_a, ANGLE_B:angle_b, ANGLE_C:angle_c, LENGTH_AB:length_ab,
LENGTH_BC:length_bc, LENGTH_CA:length_ca, DESCRIPTION:description};
    }
}

/**
 * Translate a POINT object's coordinate pair to its corresponding HTML canvas coordinates
(which are each a nonnegative integer number of pixels)
 * such that the POINT object can be graphically depicted as a two-dimensional Cartesian grid
"spaceless" location precisely located on that grid
 * (displayed inside of an HTML5 canvas element on the corresponding web page graphical
user interface).
 *
 * Assume that the relevant canvas element is square shaped and has a side length of exactly
750 pixels.
 *
 * @param {Object} input_POINT is assumed to be an object whose abstracted properties are
identical to objects returned by the function named POINT(X,Y).
 *
 * @param {String} canvas_id is assumed to be a sequence of text characters which represents
the identifier (id) of the relevant HTML canvas element.
 *
 * @return {Object} an array whose elements are exactly two nonnegative integers.
 */
function convert_POINT_coordinate_to_HTML_canvas_coordinate(input_POINT, canvas_id) {
    let the_canvas, canvas_width = 0, canvas_height = 0, output_canvas_coordinate_pair = [];
    let minimum_input_x_value = -100, maximum_input_x_value = 100;
    let minimum_input_y_value = -100, maximum_input_y_value = 100;
    let minimum_output_x_value = -100, maximum_output_x_value = 100;
    let minimum_output_y_value = -100, maximum_output_y_value = 100;
    let output_x_value = 0, output_y_value = 0;
    let output_origin_x_value = 0, output_origin_y_value = 0;
    const CANVAS_SIDE_LENGTH = 750;
    try {
        if (arguments.length !== 2) throw "exactly two (2) function inputs value are required.";
        if (typeof arguments[0].X !== "number") throw "input_POINT.X is required to be a Number
type value.";
        if (typeof arguments[0].Y !== "number") throw "input_POINT.Y is required to be a Number
type value.";
        if (typeof arguments[1] !== "string") throw "canvas_id is required to be a String type value.";
        if (Math.floor(input_POINT.X) !== input_POINT.X) throw "input_POINT.X is required to be a
whole number (i.e. integer) value.";
        if (Math.floor(input_POINT.Y) !== input_POINT.Y) throw "input_POINT.Y is required to be a
whole number (i.e. integer) value.";
        if ((input_POINT.X  maximum_input_x_value)) throw "input_POINT.X must be an integer
value inside the set [-100,100].";
        if ((input_POINT.Y  maximum_input_y_value)) throw "input_POINT.Y must be an integer
value inside the set [-100,100].";
        if (canvas_id.length  0) output_x_value = (output_origin_x_value + (input_POINT.X *
(canvas_width / (Math.abs(minimum_input_x_value) + Math.abs(maximum_input_x_value)))));
        // Determine whether or not the input_POINT is located on the top side of the x-axis of a
Cartesian plane.
        if (input_POINT.Y > 0) output_y_value = (output_origin_y_value - (input_POINT.Y *
(canvas_height / (Math.abs(minimum_input_y_value) + Math.abs(maximum_input_y_value)))));
        // Determine whether or not the input_POINT is located on the right side of the y-axis of a
Cartesian plane.
        if (input_POINT.X < 0) output_x_value = output_origin_x_value -
(Math.abs(input_POINT.X) * (canvas_width / (Math.abs(minimum_input_x_value) +
Math.abs(maximum_input_x_value))));
        // Determine whether or not the input_POINT is located on the top side of the x-axis of a
Cartesian plane.
        if (input_POINT.Y < 0) output_y_value = output_origin_y_value +
(Math.abs(input_POINT.Y) * (canvas_width / (Math.abs(minimum_input_y_value) +
Math.abs(maximum_input_y_value))));
        output_canvas_coordinate_pair.push(output_x_value);
        output_canvas_coordinate_pair.push(output_y_value);
        return output_canvas_coordinate_pair;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_POINT, canvas_id): " +
exception);
        return 0;
    }
```

```
}

/**
 * Plot a red pixel-sized dot on the canvas whose identifier (id) is "cartesian_plane" on the web
page named triangle_graphing.html
 * such that the horizontal position of the red pixel visually depicts the x-value coordinate value
represented by input_POINT
 * and such that the vertical position of the red pixel visually depicts the y-value cordinate value
represented by input_POINT
 *
 * @param {Object} input_POINT is assumed to be an object whose abstracted properties are
identical to objects returned by the function named POINT(X,Y).
 */
function plot_red_POINT_pixel_on_canvas(input_POINT) {
    let canvas, context;
    let output_canvas_coordinate_pair = [];
    try {
        canvas = document.getElementById("cartesian_plane");
        context = canvas.getContext("2d");
        output_canvas_coordinate_pair =
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_POINT, "cartesian_plane");
        context.beginPath();
        context.rect(output_canvas_coordinate_pair[0], output_canvas_coordinate_pair[1], 1, 1); //
1 pixel has a width of 1 and a height of 1
        context.strokeStyle = "#ff0000"; // HTML color code for red
        context.stroke();
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
plot_red_POINT_pixel_on_canvas(input_POINT): " + exception);
        return 0;
    }
}

/**
 * Draw a red line segment which is one pixel thick on the canvas whose identifier (id) is
"cartesian_plane" on the web page named triangle_graphing.html.
 *
 * @param {Object} input_POINT_0 is assumed to be an object whose abstracted properties are
identical to objects returned by the function named POINT(X,Y).
 *
 * @param {Object} input_POINT_1 is assumed to be an object whose abstracted properties are
identical to objects returned by the function named POINT(X,Y).
 */
```

```
/* function commented out on 21_JULY_2021
function draw_red_line_segment_on_canvas(input_POINT_0, input_POINT_1) {
    let canvas, context;
    let placeholder_array = [], ip0x = 0, ip0y = 0, ip1x = 0, ip1y = 0;
    try {
        placeholder_array =
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_POINT_0, "cartesian_plane");
        ip0x = placeholder_array[0];
        ip0y = placeholder_array[1];
        placeholder_array =
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_POINT_1, "cartesian_plane");
        ip1x = placeholder_array[0];
        ip1y = placeholder_array[1];
        canvas = document.getElementById("cartesian_plane");
        context = canvas.getContext("2d");
        context.strokeStyle = "#ff0000";
        context.lineWidth = 1;
        context.beginPath();
        context.moveTo(ip0x, ip0y);
        context.lineTo(ip1x, ip1y);
        context.stroke();
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in
draw_red_line_segment_on_canvas(input_POINT_0, input_POINT_1): " + exception);
    }
}
*/

/**
 * Draw a "light green" filled triangle which visually represents input_TRIANGLE on the canvas
whose identifier (id) is "cartesian_plane"
 * on the web page named triangle_graphing.html.
 *
 * @param {Object} input_TRIANGLE is assumed to be an object whose abstracted properties
are identical to objects returned by the function named TRIANGLE(A,B,C).
 */
function draw_green_filled_triangle(input_TRIANGLE) {
    let canvas, context;
    let placeholder_array = [], ax = 0, ay = 0, bx = 0, by = 0, cx = 0, cy = 0;
    try {
        placeholder_array =
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_TRIANGLE.A,
"cartesian_plane");
```

```
        ax = placeholder_array[0];
        ay = placeholder_array[1];
        placeholder_array =
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_TRIANGLE.B,
"cartesian_plane");
        bx = placeholder_array[0];
        by = placeholder_array[1];
        placeholder_array =
convert_POINT_coordinate_to_HTML_canvas_coordinate(input_TRIANGLE.C,
"cartesian_plane");
        cx = placeholder_array[0];
        cy = placeholder_array[1];
        canvas = document.getElementById("cartesian_plane");
        context = canvas.getContext("2d");
        context.fillStyle = "#c2fab4"; // HTML color code for a particular shade of "light green"
        context.beginPath();
        context.moveTo(ax, ay); // point 0
        context.lineTo(bx, by); // point 1
        context.lineTo(cx, cy); // point 2
        context.closePath(); // go back to point 0
        context.strokeStyle = "#ff0000"; // red
        context.lineWidth = 1;
        context.fill();
        context.stroke();
    }
    catch(exception) {
        console.log("An exception to expected functioning occurred in
draw_red_line_segment_on_canvas(input_POINT_0, input_POINT_1): " + exception);
    }
}

/**
 * Respond to the event of the GENERATE button being clicked.
 */
function generate_triangle_using_input_coordinates() {
    let cartesian_plane_canvas = "";
    let A = {}, B = {}, C = {}, T = {};
    let time_stamped_message = "", selected_menu_option_value = 0, x_coordinate_value = 0,
y_coordinate_value = 0;
    let output_div = undefined, events_log_div = undefined,
generate_button_container_paragraph = undefined;
    let select_menu_container_paragraph = undefined;
    let reset_button = undefined;
    try {
        // Append the bottom of the content inside of the "event_log" div with a time stamped
message indicating that this function was called.
        time_stamped_message = ("The function named
generate_triangle_using_input_coordinates() was called at time: " + generate_time_stamp());
        console.log(time_stamped_message);
        time_stamped_message = generate_paragraph_html_element(time_stamped_message);
        events_log_div = document.getElementById("events_log");
        events_log_div.innerHTML += time_stamped_message;
        // Replace the GENERATE button with a RESET button.
        generate_button_container_paragraph =
document.getElementById("generate_button_container");
        generate_button_container_paragraph.innerHTML = ('');
        // Transform the first input select menu (for A.X) into plain text displaying its selected
option.
        select_menu_container_paragraph = document.getElementById("a_x_menu_container");
        selected_menu_option_value = parseInt(get_selected_menu_option_value("a_x_menu"));
        select_menu_container_paragraph.innerHTML = ('A.X := ' + selected_menu_option_value
+ '. // horizontal position of two-dimensional POINT labeled A.');
        // Store the selected menu option in a variable to be used later as its corresponding POINT
property (as the property labeled X of the object labeled A).
        x_coordinate_value = selected_menu_option_value;
        // Transform the second input select menu (for A.Y) into plain text displaying its selected
option.
        select_menu_container_paragraph = document.getElementById("a_y_menu_container");
        selected_menu_option_value = parseInt(get_selected_menu_option_value("a_y_menu"));
        select_menu_container_paragraph.innerHTML = ('A.Y := ' + selected_menu_option_value
+ '. // vertical position of two-dimensional POINT labeled A.');
        // Store the selected menu option in a variable to be used later as its corresponding POINT
property (as the property labeled Y of the object labeled A).
        y_coordinate_value = selected_menu_option_value;
        // Store an Object type value for representing the two-dimensional point labeled A.
        A = POINT(x_coordinate_value,y_coordinate_value);
        // Transform the third input select menu (for B.X) into plain text displaying its selected
option.
        select_menu_container_paragraph = document.getElementById("b_x_menu_container");
        selected_menu_option_value = parseInt(get_selected_menu_option_value("b_x_menu"));
        select_menu_container_paragraph.innerHTML = ('B.X := ' + selected_menu_option_value
+ '. // horizontal position of two-dimensional POINT labeled B.');
        // Store the selected menu option in a variable to be used later as its corresponding POINT
property (as the property labeled X of the object labeled B).
        x_coordinate_value = selected_menu_option_value;
        // Transform the fourth input select menu (for B.Y) into plain text displaying its selected
option.
        select_menu_container_paragraph = document.getElementById("b_y_menu_container");
```

```
    selected_menu_option_value = parseInt(get_selected_menu_option_value("b_y_menu"));
    select_menu_container_paragraph.innerHTML = ('B.Y := ' + selected_menu_option_value + '. // vertical position of two-dimensional POINT labeled B.');
    // Store the selected menu option in a variable to be used later as its corresponding POINT property (as the property labeled Y of the object labeled B).
    y_coordinate_value = selected_menu_option_value;
    // Store an Object type value for representing the two-dimensional point labeled A.
    B = POINT(x_coordinate_value,y_coordinate_value);
    // Transform the fifth input select menu (for C.X) into plain text displaying its selected option.
    select_menu_container_paragraph = document.getElementById("c_x_menu_container");
    selected_menu_option_value = parseInt(get_selected_menu_option_value("c_x_menu"));
    select_menu_container_paragraph.innerHTML = ('C.X := ' + selected_menu_option_value + '. // horizontal position of two-dimensional POINT labeled C.');
    // Store the selected menu option in a variable to be used later as its corresponding POINT property (as the property labeled X of the object labeled C).
    x_coordinate_value = selected_menu_option_value;
    // Transform the sixth input select menu (for C.Y) into plain text displaying its selected option.
    select_menu_container_paragraph = document.getElementById("c_y_menu_container");
    selected_menu_option_value = parseInt(get_selected_menu_option_value("c_y_menu"));
    select_menu_container_paragraph.innerHTML = ('C.Y := ' + selected_menu_option_value + '. // vertical position of two-dimensional POINT labeled C.');
    // Store the selected menu option in a variable to be used later as its corresponding POINT property (as the property labeled Y of the object labeled C).
    y_coordinate_value = selected_menu_option_value;
    // Store an Object type value for representing the two-dimensional point labeled A.
    C = POINT(x_coordinate_value,y_coordinate_value);
    // Generate a TRIANGLE object using the POINT objects A, B, and C as the inputs to that "constructor" function.
    T = TRIANGLE(A,B,C);
    // Print the attributes of the TRIANGLE object as text inside of the div element whose id is "output". (Append those paragraphs to the bottom of the content in the output div).
    output_div = document.getElementById("output");
    output_div.innerHTML = generate_paragraph_html_element("T.DESCRIPTION() := " + T.DESCRIPTION() + ".");
    output_div.innerHTML += generate_paragraph_html_element("T.LENGTH_AB() := " + T.LENGTH_AB() + ". // in Cartesian grid unit lengths");
    output_div.innerHTML += generate_paragraph_html_element("T.LENGTH_BC() := " + T.LENGTH_BC() + ". // in Cartesian grid unit lengths");
    output_div.innerHTML += generate_paragraph_html_element("T.LENGTH_CA() := " + T.LENGTH_CA() + ". // in Cartesian grid unit lengths");
    output_div.innerHTML += generate_paragraph_html_element("T.PERIMETER() := " + T.PERIMETER() + ". // in Cartesian grid unit lengths");
    output_div.innerHTML += generate_paragraph_html_element("T.ANGLE_A() := " +
T.ANGLE_A() + ". // in degrees (non-obtuse between CA and AB)");
    output_div.innerHTML += generate_paragraph_html_element("T.ANGLE_B() := " +
T.ANGLE_B() + ". // in degrees (non-obtuse between AB and BC)");
    output_div.innerHTML += generate_paragraph_html_element("T.ANGLE_C() := " +
T.ANGLE_C() + ". // in degrees (non-obtuse between BC and CA)");
    output_div.innerHTML += generate_paragraph_html_element("((T.ANGLE_A() +
T.ANGLE_B()) + T.ANGLE_C()) = " + ((T.ANGLE_A() + T.ANGLE_B()) + T.ANGLE_C())  + ". // in
degrees");
    output_div.innerHTML += generate_paragraph_html_element("T.AREA() := " + T.AREA() +
". // in Cartesian grid unit square areas");
    /*
    console.log("testing POINT_coordinate_to_HTML_canvas_coordinate(input_POINT,
canvas_id)...");
    console.log('POINT_coordinate_to_HTML_canvas_coordinate(POINT(-20,-20),
"cartesian_plane") := ' + POINT_coordinate_to_HTML_canvas_coordinate(POINT(-20,-20),
"cartesian_plane") + '.');
    console.log("testing plot_red_POINT_pixel_on_canvas(POINT(-20,-20))...");
    plot_red_POINT_pixel_on_canvas(POINT(-20,-20));
    */
    // Plot the three points of the TRIANGLE object as red pixel-sized dots on the canvas
element whose id is "cartesian_plane".
    plot_red_POINT_pixel_on_canvas(T.A);
    plot_red_POINT_pixel_on_canvas(T.B);
    plot_red_POINT_pixel_on_canvas(T.C);
    //...
    console.log("testing draw_red_line_segment_on_canvas(input_POINT_0,
input_POINT_1)...");
    console.log("testing draw_red_line_segment_on_canvas(POINT(0,0),
POINT(100,100))...");
    /* code block commented out on 21_JULY_2023
    // draw_red_line_segment_on_canvas(POINT(0,0), POINT(100,100));
    // Draw red line segments which are each one pixel thick and whose endpoints are each of
the points in the TRIANGLE object.
    draw_red_line_segment_on_canvas(T.A, T.B);
    draw_red_line_segment_on_canvas(T.B, T.C);
    draw_red_line_segment_on_canvas(T.C, T.A);
    */
    // Draw a green triangular area inside of the red line segments which were previously
drawn.
    draw_green_filled_triangle(T);
  }
  catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of generate_triangle_using_input_coordinates(): " + exception);
    }
}
```