karlina_object_winter_2023_abridged_print_pages_18_to_30

[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 website named Karlina Object dot WordPress dot Com is featured also in this document].

website_address: https://karlinaobject.wordpress.com/

The final draft version of this document was published on 21_OCTOBER_2023.

HEXIDECIMAL_COLOR_CODES

The single web page application featured in this tutorial web page substantiates a three-by-three square grid composed of nine equally-sized rectangular tiles where each tile is assigned a random background color. Each tile displays the hexidecimal color code which represents that tile's background color in the format #RRGGBB along with that tile background color's RED gradient (RR), GREEN gradient (GG), and BLUE gradient (BB). The application end user may select a color skew option from a list of options to limit the randomized color output values to be either RED skewed, GREEN skewed, BLUE skewed, or not skewed towards any primary color gradient (and skew NONE is automatically selected if no other color skew menu option is selected when the GENERATE button is clicked).

If color_skew_specification is "red", the decimal number which "RR" represents will be larger than or equal to the decimal number represented by "GG" and larger than or equal to the decimal number represented by "BB".

If color_skew_specification is "green", the decimal number which "GG" represents will be larger than or equal to the decimal number represented by "RR" and larger than or equal to the decimal number represented by "BB".

If color_skew_specification is "blue", the decimal number which "BB" represents will be larger than or equal to the decimal number represented by "RR" and larger than or equal to the decimal number represented by "GG".

If color_skew_specification is neither "red" nor "green" nor "blue", then no color skew will be applied (i.e. the color_skew_specification value will be set to "none").

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

hexidecimal_digits := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']. // The nonnegative integer value of a hexidecimal digit is sixteen to the power of the decimal array index value of that particular hexidecimal digit.

primary_color_gradient_hexidecimal_value := hexidecimal_digits[P] hexidecimal_digits[Q]. // P and Q are each nonnegative decimal integer values no larger than 15.

primary_color_gradient_decimal_value := (P * (16 ^ 1)) + (Q * (16 ^ 0)). // P and Q are each nonnegative decimal integer values no larger than 15.

maximum_value(primary_color_gradient_decimal_value) := (hexidecimal_to_decimal('F') * (16 ^ 1)) + (hexidecimal_to_decimal('F') * (16 ^ 0)) = (15 * 16) + (15 * 0) = 255.

minimum_value(primary_color_gradient_decimal_value) := (hexidecimal_to_decimal('0') * (16 ^ 1)) + (hexidecimal_to_decimal('0') * (16 ^ 0)) = (0 * 16) + (0 * 0) = 0.

SOFTWARE_APPLICATION_COMPONENTS

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/hexidecimal_color_codes.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_summer_2023_starter_pack/main/hexidecimal_color_codes.js

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/time_stamp.js

---

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the HEXIDECIMAL_COLOR_CODES 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 hexidecimal_color_codes.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_packk/main/hexidecimal_color_codes.html

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

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the HEXIDECIMAL_COLOR_CODES web page application. Copy the CSS code 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;
}
```

JavaScript Code

The following JavaScript (JS) code defines most of the functions (except for one function) which control the behavior of the HEXIDECIMAL_COLOR_CODES web page application. Copy the JS code from the source code file which is linked below into a text editor and save that file as hexidecimal_color_codes.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/hexidecimal_color_codes.js

---

```
/**
 * file: hexidecimal_color_codes.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Set the web page interface to its initial state.
 *
 * Display a placeholder paragraph inside of the div element whose id is "output".
 *
 * Assume that this function is called in response to the event of the corresponding web page,
 * hexidecimal_color_codes.html, being loaded by the web browser.
 */
function initialize_application() {
    try {
        const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
        let message = "The web page, hexidecimal_color_codes.html, was loaded at time: " +
time_stamp;
        console.log(message);
        document.getElementById("time_stamped_events").innerHTML = p0 + message + p1;
        document.getElementById("output").innerHTML = p0 + "This sentence will disappear when
the GENERATE button is clicked." + p1;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
initialize_application(): " + exception);
    }
}

/**
 * Return the value of the selected menu option of a select menu element.
 *
 * @param {String} select_menu_identifier is assumed to be the id of a select HTML element.
 *
 * @return {String} the value of the selected menu option.
 */
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;
        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);
    }
}

/**
 * Use the native JavaScript Math library function for generating random numbers to select a
 * base-ten number no smaller than 0 and less than 1.
 *
 * @return {Number} a base-ten (i.e. decimal) number no smaller than zero and smaller than
one.
 */
function generate_random_nonnegative_number_less_than_one() {
    return Math.random();
}
```

```
/**
 * Use the native JavaScript Math library function for generating random numbers to select a
base-ten
 * number no smaller than 0 and less than 1 and store the result in a variable named N.
 *
 * Multiply N by 256, round the result down to the nearest integer, and return that rounded down
result.
 *
 * @return {Number} a base-ten (i.e. decimal) integer no smaller than 0 and no larger than 255.
 */
function generate_random_nonnegative_integer_less_than_two_hundred_fifty_six() {
   let N = generate_random_nonnegative_number_less_than_one();
   return Math.floor(N * 256);
}

/**
 * Convert a nonnegative decimal integer which is no smaller than 0 and no larger than 255 to
its two-digit hexidecimal (i.e. base-sixteen) equivalent.
 *
 * @param {Number} decimal_integer is assumed to be a base-ten integer no smaller than 0
and no larger than 255.
 *
 * @return {String} a sequence of two hexidecimal digits which represents the same numerical
value as decimal_integer.
 */
function convert_from_decimal_to_hexidecimal(decimal_integer) {
   try {
      let hexidecimal_digits_set = "0123456789ABCDEF";
      let hexidecimal_output_array = ['0','0'];
      let hexidecimal_output_string = "";
      if (arguments.length !== 1) throw "exactly one function argument is required.";
      if (typeof arguments[0] !== "number") throw "decimal_integer must be a Number type
value.";
      if (decimal_integer !== Math.floor(decimal_integer)) throw "decimal_integer must be a
whole number value.";
      if (decimal_integer < 0) throw "decimal_integer must be a nonnegative number value.";
      if (decimal_integer > 255) throw "decicmal_integer must not exceed 255.";
      hexidecimal_output_array[0] = hexidecimal_digits_set[decimal_integer % 16];
      decimal_integer = Math.floor(decimal_integer / 16);
      hexidecimal_output_array[1] = hexidecimal_digits_set[decimal_integer % 16];
      hexidecimal_output_string += hexidecimal_output_array[1];
      hexidecimal_output_string += hexidecimal_output_array[0];
      return hexidecimal_output_string;
   }
```

```
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_from_decimal_to_hexidecimal(decimal_integer): " + exception);
        return "00";
    }
}

/**
 * Generate a random hexidecimal color code string in the format "#RRGGBB"
 * such that RR represents the RED gradient,
 * GG represents the GREEN gradient, and
 * BB represents the BLUE gradient
 * (and such that each of the three "primary color" gradients
 * is a sequence of two hexidecimal digits which represents a decimal integer
 * no smaller than 0 and no larger than 255)).
 *
 * If color_skew_specification is "red",
 * the decimal number which "RR" represents will be
 * larger than or equal to the decimal number represented by "GG" and
 * larger than or equal to the decimal number represented by "BB".
 *
 * If color_skew_specification is "green",
 * the decimal number which "GG" represents will be
 * larger than or equal to the decimal number represented by "RR" and
 * larger than or equal to the decimal number represented by "BB".
 *
 * If color_skew_specification is "blue",
 * the decimal number which "BB" represents will be
 * larger than or equal to the decimal number represented by "RR" and
 * larger than or equal to the decimal number represented by "GG".
 *
 * If color_skew_specification is neither "red" nor "green" nor "blue",
 * then no color skew will be applied (i.e. the color_skew_specification value will be set to
"none").
 *
 * Note that a hexidecimal digit is a String type character from the following set of sixteen
base-sixteen digits
 * (which are listed in ascending order such that each hexidecimal digit
 * represents a base-ten integer value starting with 0 and incrementing by exactly 1):
 *
 * hexidecimal_digits := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'].
 *
 * @param {String} color_skew_specification is assumed to be either "none", "red", "green", or
"blue".
```

```
 *
 * @return {String} a hexidecimal color code string in the format "#RRGGBB".
 */
function generate_random_hexidecimal_color_code(color_skew_specification) {
   try {
      let red_gradient_hexidecimal_digits_sequence = "00", red_gradient_decimal_value = 0;
      let green_gradient_hexidecimal_digits_sequence = "00", green_gradient_decimal_value =
0;
      let blue_gradient_hexidecimal_digits_sequence = "00", blue_gradient_decimal_value = 0;
      if (arguments.length !== 1) throw "exactly one function argument is required.";
      if (typeof arguments[0] !== "string") throw "color_skew_specification must be a String type
value.";
      red_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
      green_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
      blue_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
      if (color_skew_specification === "red") {
         while ((red_gradient_decimal_value < green_gradient_decimal_value) ||
(red_gradient_decimal_value < blue_gradient_decimal_value)) {
            red_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
            green_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
            blue_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
         }
      }
      if (color_skew_specification === "green") {
         while ((green_gradient_decimal_value < red_gradient_decimal_value) ||
(green_gradient_decimal_value < blue_gradient_decimal_value)) {
            red_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
            green_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
            blue_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
         }
      }
      if (color_skew_specification === "blue") {
         while ((blue_gradient_decimal_value < red_gradient_decimal_value) ||
(blue_gradient_decimal_value < green_gradient_decimal_value)) {
```

```
                red_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
                green_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
                blue_gradient_decimal_value =
generate_random_nonnegative_integer_less_than_two_hundred_fifty_six();
            }
        }
        red_gradient_hexidecimal_digits_sequence =
convert_from_decimal_to_hexidecimal(red_gradient_decimal_value);
        green_gradient_hexidecimal_digits_sequence =
convert_from_decimal_to_hexidecimal(green_gradient_decimal_value);
        blue_gradient_hexidecimal_digits_sequence =
convert_from_decimal_to_hexidecimal(blue_gradient_decimal_value);
        return "#" + red_gradient_hexidecimal_digits_sequence +
green_gradient_hexidecimal_digits_sequence + blue_gradient_hexidecimal_digits_sequence;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
generate_random_hexidecimal_color_code(color_skew_specification): " + exception);
        return "#000000";
    }
}

/**
 * Return the decimal integer index of the hexidecimal digit which H represents in the array
below:
 *
 * hexidecimal_digits := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'].
 *
 * @param {String} H is assumed to be a single hexidecimal digit.
 *
 * @return {Number} the decimal integer index of the hexidecimal digit which H represents.
 */
function get_decimal_value_of_hexidecimal_digit(H) {
    if (H === '0') return 0;
    if (H === '1') return 1;
    if (H === '2') return 2;
    if (H === '3') return 3;
    if (H === '4') return 4;
    if (H === '5') return 5;
    if (H === '6') return 6;
    if (H === '7') return 7;
    if (H === '8') return 8;
    if (H === '9') return 9;
    if (H === 'A') return 10;
    if (H === 'B') return 11;
    if (H === 'C') return 12;
    if (H === 'D') return 13;
    if (H === 'E') return 14;
    if (H === 'F') return 15;
    return 0;
}

/**
 * Convert a string comprised of exactly two hexidecimal digits to its corresponding decimal
representation.
 *
 * Note that the characters comprising hexidecimal_sequence must be members of the following
set:
 *
 * hexidecimal_digits := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'].
 *
 * Note that the output value will be a decimal integer no smaller than 0 and no larger than 255.
 *
 * @param {String} hexidecimal_sequence is assumed to be a string comprised of exactly two
hexidecimal digits.
 *
 * @return {Number} the base-ten number which hexidecimal_sequence represents.
 */
function convert_from_hexidecimal_to_decimal(hexidecimal_sequence) {
    try {
        let hexidecimal_digits_set = "0123456789ABCDEF", i = 0, is_valid_hexidecimal_digit =
false, decimal_output = 0;
        if (arguments.length !== 1) throw "exactly one function argument is required.";
        if (typeof arguments[0] !== "string") throw "hexidecimal_sequence must be a String type
value.";
        if (hexidecimal_sequence.length !== 2) throw "hexidecimal_sequence must have a length
of exactly 2 characters.";
        for (i = 0; i < 16; i += 1) if (hexidecimal_sequence[0] === hexidecimal_digits_set[i])
is_valid_hexidecimal_digit = true;
        if (!is_valid_hexidecimal_digit) throw "The first digit of hexidecimal_sequence is not a valid
hexidecimal digit.";
        is_valid_hexidecimal_digit = false;
        for (i = 0; i < 16; i += 1) if (hexidecimal_sequence[1] === hexidecimal_digits_set[i])
is_valid_hexidecimal_digit = true;
        if (!is_valid_hexidecimal_digit) throw "The second digit of hexidecimal_sequence is not a
valid hexidecimal digit.";
```

```
        decimal_output += (get_decimal_value_of_hexidecimal_digit(hexidecimal_sequence[0]) *
16);
        decimal_output += (get_decimal_value_of_hexidecimal_digit(hexidecimal_sequence[1]));
        return decimal_output;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_from_hexidecimal_to_decimal(hexidecimal_sequence): " + exception);
        return 0;
    }
}

/**
 * Generate the HTML string which constructs a table data cell whose text content is as follows:
 *
 * - hexidecimal_color_code: a random hexidecimal color code string in the format "#RRGGBB"
 *                           such that RR represents the RED gradient,
 *                           GG represents the GREEN gradient, and
 *                           BB represents the BLUE gradient
 *                           (and such that each of the three "primary color" gradients
 *                           is a sequence of two hexidecimal digits which represents a decimal integer
 *                           no smaller than 0 and no larger than 255)).
 *
 * - red_value: a decimal integer no smaller than 0 and no larger than 255 which "RR"
represents.
 *
 * - green_value: a decimal integer no smaller than 0 and no larger than 255 which "GG"
represents.
 *
 * - blue_value: a decimal integer no smaller than 0 and no larger than 255 which "BB"
represents.
 *
 * If color_skew_specification is "red",
 * the decimal number which "RR" represents will be larger than or equal to the decimal number
represented by "GG"
 * and larger than or equal to the decimal number represented by "BB".
 *
 * If color_skew_specification is "green",
 * the decimal number which "GG" represents will be larger than or equal to the decimal number
represented by "RR"
 * and larger than or equal to the decimal number represented by "BB".
 *
 * If color_skew_specification is "blue",
 * the decimal number which "BB" represents will be larger than or equal to the decimal number
represented by "RR"
 * and larger than or equal to the decimal number represented by "GG".
 *
 * If color_skew_specification is neither "red" nor "green" nor "blue",
 * then no color skew will be applied (i.e. the color_skew_specification value will be set to
"none").
 *
 * @param {String} color_skew_specification is assumed to be either "none", "red", "green", or
"blue".
 *
 * @return {String} the HTML string which may be used to construct a table data element on a
web page.
 */
function generate_html_string_for_table_cell(color_skew_specification) {
   let html_string = "", hexidecimal_color_code = "", RR = "", GG = "", BB = "", red_value = 0,
green_value = 0, blue_value = 0;
   hexidecimal_color_code =
generate_random_hexidecimal_color_code(color_skew_specification);
   RR += hexidecimal_color_code[1];
   RR += hexidecimal_color_code[2];
   GG += hexidecimal_color_code[3];
   GG += hexidecimal_color_code[4];
   BB += hexidecimal_color_code[5];
   BB += hexidecimal_color_code[6];
   red_value = convert_from_hexidecimal_to_decimal(RR);
   green_value = convert_from_hexidecimal_to_decimal(GG);
   blue_value = convert_from_hexidecimal_to_decimal(BB);
   html_string += '<' + 'td' + ' style="background:' +  hexidecimal_color_code + '"' + '>';
   html_string += '<' + 'p' + ' class="tile_text"' + '>';
   html_string += "hexidecimal_color_code: " + hexidecimal_color_code;
   html_string += '<' + '/' + 'p' + '>';
   html_string += '<' + 'p' + ' class="tile_text"' + '>';
   html_string += "red_value: " + red_value;
   html_string += '<' + '/' + 'p' + '>';
   html_string += '<' + 'p' + ' class="tile_text"' + '>';
   html_string += "green_value: " + green_value;
   html_string += '<' + '/' + 'p' + '>';
   html_string += '<' + 'p' + ' class="tile_text"' + '>';
   html_string += "blue_value: " + blue_value;
   html_string += '<' + '/' + 'p' + '>';
   html_string += '<' + '/' + 'td' + '>';
   return html_string;
}
```

```
/**
 * Populate the div element whose id is "output" with a table consisting of three rows, three
columns, and data cells
 * such that the background color of each data cell is set to the hexidecimal color code text
 * displayed inside of that particular data cell.
 *
 * Each of the nine hexidecimal color codes is a randomly generated String value whose
RED-GREEN-BLUE values are
 * constrained according to the option selected in the select menu whose id is
"color_skew_menu".
 *
 * Assume that this function is called in response to the event of button on the corresponding
web page,
 * hexidecimal_color_codes.html, whose value is "GENERATE" being clicked.
 */
function generate_color_grid() {
   try {
      const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
      let message = "The GENERATE button was clicked at time: " + time_stamp;
      let selected_color_skew = "", output_div_content = "";
      console.log(message);
      document.getElementById("time_stamped_events").innerHTML += p0 + message + p1;
      selected_color_skew = get_selected_menu_option_value("color_skew_menu");
      output_div_content = '<' + 'p' + '>';
      output_div_content += "selected_color_skew := " + selected_color_skew + ".";
      output_div_content += '<' + '/' + 'p' + '>';
      output_div_content += '<' + 'table' + '>';
      output_div_content += '<' + 'tr' + '>';
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += '<' + '/' + 'tr' + '>';
      output_div_content += '<' + 'tr' + '>';
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += '<' + '/' + 'tr' + '>';
      output_div_content += '<' + 'tr' + '>';
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += generate_html_string_for_table_cell(selected_color_skew);
      output_div_content += '<' + '/' + 'tr' + '>';
      output_div_content += '<' + '/' + 'table' + '>';
```

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

JavaScript Code

The following JavaScript (JS) code defines one function which controls the behavior of the HEXIDECIMAL_COLOR_CODES web page application; namely the function which returns the number of milliseconds elapsed since the Unix Epoch concatenated to the string ” milliseconds since midnight on 01_JANUARY_1970″. Copy the JS code from the source code file which is linked below into a text editor and save that file as time_stamp.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/time_stamp.js

---

```
/**
 * file: time_stamp.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Get the Number of milliseconds which have elapsed since the Unix Epoch.
 * The Unix Epoch is 01_JANUARY_1970 at midnight (Coordinated Universal Time (UTC)).
 *
 * @return {String} message displaying the time at which this function was called.
 */
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.";
}
```

HEXIDECIMAL_COLOR_CODES Interface (Initial)

The screenshot image below depicts what the HEXIDECIMAL_COLOR_CODES web page interface is supposed to look like when the web page is initially loaded by a web browser.

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

---

HEXIDECIMAL_COLOR_CODES Interface (Skew None)

The screenshot image below depicts one possible configuration of what the HEXIDECIMAL_COLOR_CODES web page interface could look like if the GENERATE button is clicked without first clicking on the color skew menu button or if the "NONE" option is selected in the color skew menu before the GENERATE button is clicked.

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

---

HEXIDECIMAL_COLOR_CODES Interface (Skew Red)

The screenshot image below depicts one possible configuration of what the HEXIDECIMAL_COLOR_CODES web page interface could look like if the "RED" option is selected in the color skew menu before the GENERATE button is clicked.

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

---

HEXIDECIMAL_COLOR_CODES Interface (Skew Green)

The screenshot image below depicts one possible configuration of what the HEXIDECIMAL_COLOR_CODES web page interface could look like if the "GREEN" option is selected in the color skew menu before the GENERATE button is clicked.

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

---

HEXIDECIMAL_COLOR_CODES Interface (Skew Blue)

The screenshot image below depicts one possible configuration of what the HEXIDECIMAL_COLOR_CODES web page interface could look like if the "BLUE" option is selected in the color skew menu before the GENERATE button is clicked.

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

---

This web page was last updated on 10_JULY_2023. The content displayed on this web page is licensed as PUBLIC_DOMAIN intellectual property.

[End of abridged plain-text content from HEXIDECIMAL_COLOR_CODES]

BITS_AND_BYTES

The single web page application featured in this tutorial web page allows the end user to click switches which each flip a single uniquely corresponding light bulb from "OFF" to "ON" and vice versa and, also, to click a BINARY_TO_DECIMAL button which computes the binary number represented by the light bulb array configuration at the time that button is clicked. Algebraic equations which are used to convert the base-two integer value represented by the sequence of eight light bulbs to that base-two integer's base-ten equivalent value are displayed at the bottom of this web page. (Clicking the RESET button will return this web page to its initial state such that each one of the eight light bulbs is set to "OFF" and all buttons on this web page are set to visible).

The binary integer value 0 is an abstract representation of the relatively physical "OFF" (insufficiently high voltage) light bulb state.

The binary integer value 1 is an abstract representation of the relatively physical "ON" (sufficiently high voltage) light bulb state.

Each one of the eight light bulbs displayed in the table below represents a particular power of two.

(N in the table below represents a nonnegative integer value no larger than seven).

While light_bulb_N is in its “ON” state, light_bulb_N represents two multiplied by itself N times.

While light_bulb_N is in its “OFF” state, light_bulb_N represents zero.

The nonnegative decimal integer value which the sequence of eight light bulbs represents is computed by adding up the nonnegative decimal integer values which each of those eight light bulbs represents.

The largest decimal integer value which the sequence of eight light bulbs can represent is 255.

The smallest decimal integer value which the sequence of eight light bulbs can represents is 0.

The total number of unique states which the sequence of eight light bulbs can represent is 256.

The largest binary integer value which any one of the eight light bulbs can represent is 1.

The smallest binary integer value which any one of the eight light bulbs can represent is 0.

The total number of unqiue states which any one of the eight light bulbs can represent is 2.

The rightmost light bulb has an N value of 0.

The rightmost light bulb represents the “lowest order bit” in the byte which the sequence of eight light bulb represents.

lowest_order_bit(ON) = 1 * (2 ^ 0) = 1 * 1 = 1.

lowest_order_bit(OFF) = 0 * (2 ^ 0) = 0 * 1 = 0.

The leftmost light bulb has an N value of 7.

The leftmost light bulb represents the “highest order bit” in the byte which the sequence of eight light bulb represents.

highest_order_bit(ON) = 1 * (2 ^ 7) = 1 * 2 * 2 * 2 * 2 * 2 * 2 * 2 = 128.

highest_order_bit(OFF) = 0 * (2 ^ 7) = 0 * 2 * 2 * 2 * 2 * 2 * 2 * 2 = 0.

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

BIT := abbreviation("binary digit") = OR(0,1). // A bit represents one of two possible unique states at a time (and those two states are the integers 0 and 1).

BYTE := abbreviation("binary term") = BIT[8]. // A byte represents a sequence of eight bits.

SOFTWARE_APPLICATION_COMPONENTS

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/binary_to_decimal.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_summer_2023_starter_pack/main/binary_to_decimal.js

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

image_link:
https://raw.githubusercontent.com/KARLINA_OBJECT_summer_2023_starter_pack/main/light_bulb_on.png

---

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the BINARY_TO_DECIMAL 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 binary_to_decimal.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/binary_to_decimal.html

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

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the BINARY_TO_DECIMAL web page application. Copy the CSS code 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;
```

```
}
```

JavaScript Code

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

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/binary_to_decimal.js

---

```
/**
 * file: binary_to_decimal.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Display a time stamped message inside of the div element whose id is "output".
 *
 * Set each of the eight light bulbs to "OFF" rather rather than to "ON".
 *
 * Set each of the eight binary digit values to zero (0) rather than to one (1).
 *
 * Set each of the buttons on the web page to visible rather than to hidden.
 */
function initialize_application() {
        try {
        const time_point = Date.now(), p0 = ('<' + 'p' + '>'), p1 = ('<' + '/' + 'p' + '>');
        let light_bulb_off = '<' + 'img src="light_bulb_off.png" width="60"' + '>', i = 0;
        const message = "The initialize_application() function was called at time: " + time_point
+ " milliseconds since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).";
        console.log(message);
        document.getElementById("output").innerHTML = p0 + message + p1;
        for (i = 0; i < 8; i += 1) {
        document.getElementById("light_bulb_" + i).innerHTML = light_bulb_off;
        document.getElementById("bit_" + i).innerHTML = 0;
```

```
          document.getElementById("switch_" + i).style.display = "block";
          }
          document.getElementById("binary_to_decimal_button").style.display = "inline";
          }
          catch(exception) {
          console.log("An exception to expected functioning occurred during the runtime of the
JavaScript function named initialize_application(): " + exception);
          }
}

/**
 * Change the binary state represented by the Nth light bulb in the sequence of 8 light bulbs to
its only alternative state.
 *
 * If the Nth bit value is set to 0, the Nth light bulb state is assumed to be "OFF".
 *
 * If the Nth bit value is set to 1, the Nth light bulb state is assumed to be "ON".
 *
 * When the switch for the Nth light bulb is clicked, the following will occur:
 *
 * If the Nth bit is set to 0, the Nth bit will be set to 1 and the Nth light bulb state will be set to
"ON".
 *
 * If the Nth bit is set to 1, the Nth bit will be set to 0 and the Nth light bulb state will be set to
"OFF".
 *
 * @param {Number} N is assumed to be a base-ten integer no smaller than 0 and no larger
than 7.
 */
function binary_switch(N) {
          try {
          const p0 = ('<' + 'p' + '>'), p1 = ('<' + '/' + 'p' + '>');
          let bit = undefined, light_bulb_image = undefined, light_bulb_off = '<' + 'img
src="light_bulb_off.png" width="60"' + '>', light_bulb_on = '<' + 'img src="light_bulb_on.png"
width="60"' + '>';
          bit = parseInt(document.getElementById("bit_" + N).innerHTML);
          if (bit === 0) {
          document.getElementById("bit_" + N).innerHTML = 1;
          document.getElementById("light_bulb_" + N).innerHTML = light_bulb_on;
          }
          else {
          document.getElementById("bit_" + N).innerHTML = 0;
          document.getElementById("light_bulb_" + N).innerHTML = light_bulb_off;
          }
	}
	catch(exception) {
	console.log("An exception to expected functioning occurred during the runtime of the
JavaScript function named binary_switch(N): " + exception);
	}
}

/**
 * Change the binary state represented by the 7th (i.e. seventh) light bulb in the array of 8 light
bulbs to its only alternative state.
 *
 * Note that the 7th light bulb represents the highest order bit and is the leftmost light bulb in the
array.
 *
 * If the 7th bit value is set to 0, the 7th light bulb state is assumed to be "OFF".
 *
 * If the 7th bit value is set to 1, the 7th light bulb state is assumed to be "ON".
 *
 * When the switch for the 7th light bulb is clicked, the following will occur:
 *
 * If the 7th bit is set to 0, then the Nth bit will be set to 1 and the 7th light bulb state will be set to
"ON".
 *
 * Otherwise, the 7th bit will be set to 0 and the 7th light bulb state will be set to "OFF".
 */
function switch_7() {
	const time_point = Date.now();
	console.log("The switch_7 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(7);
}

/**
 * Change the binary state represented by the 6th (i.e. sixth) light bulb in the array of 8 light
bulbs to its only alternative state.
 *
 * If the 6th bit value is set to 0, the 6th light bulb state is assumed to be "OFF".
 *
 * If the 6th bit value is set to 1, the 6th light bulb state is assumed to be "ON".
 *
 * When the switch for the 6th light bulb is clicked, the following will occur:
 *
 * If the 6th bit is set to 0, then the 6th bit will be set to 1 and the 6th light bulb state will be set to
"ON".
```

```
 *
 * Otherwise, the 6th bit will be set to 0 and the 6th light bulb state will be set to "OFF".
 */
function switch_6() {
	const time_point = Date.now();
	console.log("The switch_6 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(6);
}

/**
 * Change the binary state represented by the 5th (i.e. fifth) light bulb in the array of 8 light bulbs
to its only alternative state.
 *
 * If the 5th bit value is set to 0, the 5th light bulb state is assumed to be "OFF".
 *
 * If the 5th bit value is set to 1, the 5th light bulb state is assumed to be "ON".
 *
 * When the switch for the 5th light bulb is clicked, the following will occur:
 *
 * If the 5th bit is set to 0, then the 5th bit will be set to 1 and the 5th light bulb state will be set to
"ON".
 *
 * Otherwise, the 5th bit will be set to 0 and the 5th light bulb state will be set to "OFF".
 */
function switch_5() {
	const time_point = Date.now();
	console.log("The switch_5 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(5);
}

/**
 * Change the binary state represented by the 4th (i.e. fourth) light bulb in the array of 8 light
bulbs to its only alternative state.
 *
 * If the 4th bit value is set to 0, the 4th light bulb state is assumed to be "OFF".
 *
 * If the 4th bit value is set to 1, the 4th light bulb state is assumed to be "ON".
 *
 * When the switch for the 4th light bulb is clicked, the following will occur:
 *
 * If the 4th bit is set to 0, then the 4th bit will be set to 1 and the 4th light bulb state will be set to
"ON".
```

```
 *
 * Otherwise, the 4th bit will be set to 0 and the 4th light bulb state will be set to "OFF".
 */
function switch_4() {
	const time_point = Date.now();
	console.log("The switch_4 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(4);
}

/**
 * Change the binary state represented by the 3rd (i.e. third) light bulb in the array of 8 light
bulbs to its only alternative state.
 *
 * If the 3rd bit value is set to 0, the 3rd light bulb state is assumed to be "OFF".
 *
 * If the 3rd bit value is set to 1, the 3rd light bulb state is assumed to be "ON".
 *
 * When the switch for the 3rd light bulb is clicked, the following will occur:
 *
 * If the 3rd bit is set to 0, then the 3rd bit will be set to 1 and the 3rd light bulb state will be set to
"ON".
 *
 * Otherwise, the 3rd bit will be set to 0 and the 3rd light bulb state will be set to "OFF".
 */
function switch_3() {
	const time_point = Date.now();
	console.log("The switch_3 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(3);
}

/**
 * Change the binary state represented by the 2nd (i.e. second) light bulb in the array of 8 light
bulbs to its only alternative state.
 *
 * If the 2nd bit value is set to 0, the 2nd light bulb state is assumed to be "OFF".
 *
 * If the 2nd bit value is set to 1, the 2nd light bulb state is assumed to be "ON".
 *
 * When the switch for the 2nd light bulb is clicked, the following will occur:
 *
 * If the 2nd bit is set to 0, then the 2nd bit will be set to 1 and the 2nd light bulb state will be set
to "ON".
```

```
 *
 * Otherwise, the 2nd bit will be set to 0 and the 2nd light bulb state will be set to "OFF".
 */
function switch_2() {
	const time_point = Date.now();
	console.log("The switch_2 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(2);
}

/**
 * Change the binary state represented by the 1st (i.e. first) light bulb in the array of 8 light bulbs
to its only alternative state.
 *
 * If the 1st bit value is set to 0, the 1st light bulb state is assumed to be "OFF".
 *
 * If the 1st bit value is set to 1, the 1st light bulb state is assumed to be "ON".
 *
 * When the switch for the 1st light bulb is clicked, the following will occur:
 *
 * If the 1st bit is set to 0, then the 1st bit will be set to 1 and the 1st light bulb state will be set to
"ON".
 *
 * Otherwise, the 1st bit will be set to 0 and the 1st light bulb state will be set to "OFF".
 */
function switch_1() {
	const time_point = Date.now();
	console.log("The switch_1 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
	binary_switch(1);
}

/**
 * Change the binary state represented by the 0th (i.e. zeroth) light bulb in the array of 8 light
bulbs to its only alternative state.
 *
 * Note that the 0th light bulb represents the lowest order bit and is the rightmost light bulb in the
array.
 *
 * If the 0th bit value is set to 0, the 0th light bulb state is assumed to be "OFF".
 *
 * If the 0th bit value is set to 1, the 0th light bulb state is assumed to be "ON".
 *
 * When the switch for the 1st light bulb is clicked, the following will occur:
```

```
 *
 * If the 0th bit is set to 0, then the 0th bit will be set to 1 and the 1st light bulb state will be set to
"ON".
 *
 * Otherwise, the 0th bit will be set to 0 and the 0th light bulb state will be set to "OFF".
 */
function switch_0() {
        const time_point = Date.now();
        console.log("The switch_0 button was clicked at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).");
        binary_switch(0);
}

/**
 * Hide all buttons except for the RESET button.
 *
 * Convert the input binary number (represented by the current configuration of eight light bulbs)
 * to its logically equivalent decimal output number and
 * display the arithmetic steps used to convert the input value to the output value inside the div
element whose id is "output".
 */
function binary_to_decimal() {
        try {
        const time_point = Date.now(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>', s0 = '<' + 'span
class="console"' + '>', s1 = '<' + '/' + 'span' + '>';
        let i = 0, binary_digits_string = "", decimal_output_number = 0, decimal_term_value = 0,
arithmetic_steps = "", message = "";
        message = "The BINARY_TO_DECIMAL button was clicked at time: " + time_point + "
milliseconds since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).";
        console.log(message);
        document.getElementById("output").innerHTML += p0 + message + p1;
        for (i = 0; i < 8; i += 1) document.getElementById("switch_" + i).style.display = "none";
        document.getElementById("binary_to_decimal_button").style.display = "none";
        for (i = 7; i > -1; i -= 1) binary_digits_string += document.getElementById("bit_" +
i).innerHTML;
        console.log("The input binary digit sequence is " + binary_digits_string + ".");
        for (i = 7; i > -1; i -= 1) {
        decimal_term_value = parseInt(binary_digits_string[i]) * Math.pow(2, 7 - i);
        decimal_output_number += decimal_term_value;

        }
        console.log("The output decimal digit sequence is " + decimal_output_number + ".");
        arithmetic_steps += p0 + "binary_input: " + s0 + binary_digits_string + s1 + "." + p1;
        arithmetic_steps += p0 + "decimal_output: " + decimal_output_number + "." + p1;
        arithmetic_steps += p0 + "arithmetic_steps: " + p1;
        arithmetic_steps += p0 + decimal_output_number + " = " + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[0] + s1 + " * (2 ^ 7)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[1] + s1 + " * (2 ^ 6)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[2] + s1 + " * (2 ^ 5)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[3] + s1 + " * (2 ^ 4)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[4] + s1 + " * (2 ^ 3)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[5] + s1 + " * (2 ^ 2)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[6] + s1 + " * (2 ^ 1)) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[7] + s1 + " * (2 ^ 0)) = " + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[0] + s1 + " * 128) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[1] + s1 + " * 64) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[2] + s1 + " * 32) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[3] + s1 + " * 16) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[4] + s1 + " * 8) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[5] + s1 + " * 4) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[6] + s1 + " * 2) +" + p1;
        arithmetic_steps += p0 + "(" + s0 + binary_digits_string[7] + s1 + " * 1) = " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[0]) * 128 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[1]) * 64 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[2]) * 32 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[3]) * 16 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[4]) * 8 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[5]) * 4 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[6]) * 2 + " + " + p1;
        arithmetic_steps += p0 + parseInt(binary_digits_string[7]) * 1 + "." + p1;
        document.getElementById("output").innerHTML += arithmetic_steps;
        }
        catch(exception) {
        console.log("An exception to expected functioning occurred during the runtime of the
JavaScript function named binary_to_decimal(): " + exception);
        }
}
```

BINARY_TO_DECIMAL Interface (Progress)

The screenshot image below depicts what the BINARY_TO_DECIMAL web page interface looks like after some of the eight binary switches are clicked.

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starte
r_pack/main/binary_to_decimal_interface_progress.png’

BINARY_TO_DECIMAL Interface (Final)

The screenshot image below depicts what the BINARY_TO_DECIMAL web page interface looks like after some of the eight binary switches are clicked and then the BINARY_TO_DECIMAL button is clicked.

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

---

This web page was last updated on 10_JULY_2023. The content displayed on this web page is licensed as PUBLIC_DOMAIN intellectual property.

[End of abridged plain-text content from BITS_AND_BYTES]

COUNT_DOWN_TIMER

Time is what makes the observation of change possible.

Time is what prevents all phenomena from being perceived as occurring simultaneously.

The single web page application featured in this tutorial web page substantiates a count-down timer which enables the application user to select a value for N (i.e. the natural number of seconds to elapse to zero). When the application user clicks the start button, the start button will disappear and the number of seconds remaining will decrement by 1 per second starting at N until the number of seconds remaining is 0. When the number of seconds remaining is decremented to zero, an alert sound effect will be played and the start button will re-appear.

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

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the COUNT_DOWN_TIMER 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 count_down_timer.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/count_down_timer.html

Portable-Network-Graphics_image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/count_down_timer_html_code_screenshot.png

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the COUNT_DOWN_TIMER web page application. Copy the CSS code 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
 */
```

```css
/** 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;
}
```

JavaScript Code

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

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/count_down_timer.js

---

```
/**
 * file: count_down_timer.js
 * type: JavaScript
 * author: karbytes
 * date: 10_JULY_2023
 * license: PUBLIC_DOMAIN
 */

/**
 * Generate the HTML code for the portion of the web page interface display
 * which outputs the quantity of temporal units elapsed starting at
 * the time at which the start button was clicked and ending at the
 * time at which the selected number of time units is decremented
 * to zero.
 *
 * @return {String} the HTML for defining a SPAN element whose id is
"nonnegative_integer_display"
 */
function generate_nonnegative_integer_display_span() {
  return ('<' + 'span id="nonnegative_integer_display"' + '>') + '0' + ('<' + '/' + 'span' + '>');
}

/**
 * Display a time-stamped message inside the DIV element whose identifier (id)
 * is "application_status_messages_display".
 * which denotes the point in time at which the web page application was
 * either opened or else reloaded by a web browser.
 *
 * Display "0" inside the SPAN element whose identifier is "seconds_elapsed_display".
 *
 * Set the input element whose identifier is "timer_session_start_button"
 * to visible rather than to hidden.
 *
 * Assume that this function is called when a web broswer opens count_down_timer.html
 * or else reloads that web page.
 */
function initialize_application() {
  try {
          const message = ("The web page was opened at time: " + generate_time_stamp());
          console.log(message);
          document.getElementById("timer_start_button").style.visibility = "block";
          document.getElementById("seconds_elapsed_display").innerHTML = (('<' + 'p' + '>') +
"seconds_remaining: " + generate_nonnegative_integer_display_span() + ('<' + '/' + 'p' + '>'));
          document.getElementById("application_status_messages_display").innerHTML = (('<' +
'p' + '>') + message + ('<' + '/' + 'p' + '>'));
  }
  catch(e) {
          console.log("An exception to normal functioning occurred during the runtime of
initialize_application(): " + e);
  }
}

/**
 * Return the value of the selected menu OPTION of a SELECT menu element.
 *
 * @param {String} select_menu_identifier is the id of a select HTML element.
 *
 * @return {String} the value of the selected menu option.
 */
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;
```

```
          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(e) {
          console.log("An exception to normal functioning occurred during the runtime of
get_selected_menu_option(select_menu_identifier): " + e);
  }
}

/**
 * Play exactly one sound file exactly one time
 * (i.e. for a finite duration of time there will
 * be a sequence of auditory phenomena).
 *
 * Assume that the sound file, alert_sound_effect.wav,
 * exists in the same file directory as this JavaScript file(s)
 * and the web page deploying this JavaScript file(s) and the
 * referenced audio file.
 */
function play_sound_file() {
  try {
          var audio = new Audio("alert_sound_effect.wav");
          audio.play();
  }
  catch(e) {
          console.log("An exception to normal functioning occurred during the runtime of
play_sound_file(): " + e);
  }
}

/**
 * Terminate the timer session.
 *
 * Assume that this function is called after the
 * decrement_timer_display_by_one_second(simulation)
 * function is called N times.
 *
 * Print a time-stamped status message to the DIV element
 * whose id is "application_status_messages_display"
 * which indicates that the timer session of N seconds
 * is finished.
 *
 * Attempt to play a sound which signifies that the timer
 * session has completed.
 */
function finish_simulation() {
  try {
        const message = ("The timed simulation finished at time: " + generate_time_stamp());
        console.log(message);
        document.getElementById("application_status_messages_display").innerHTML += (('<'
+ 'p' + '>') + message + ('<' + '/' + 'p' + '>'));
        document.getElementById("timer_start_button").style.display = "block";
        play_sound_file();
  }
  catch(e) {
        console.log("An exception to normal functioning occurred during the runtime of
finish_simulation(): " + e);
  }
}

/**
 * Decrement the number of seconds remaining by one second.
 * If the number of seconds is less than one, then terminate the timer session.
 *
 * Assume that there is a text HTML element whose id is "seconds_elapsed_display"
 * (and that the inner HTML of that element is an integer value).
 *
 * @param {Object} simulation - an instance of some temporally finite process
 */
function decrement_timer_display_by_one_second(simulation) {
  try {
        let timer_display, seconds_remaining;
        timer_display = document.getElementById("seconds_elapsed_display");
        seconds_remaining = parseInt(timer_display.innerHTML);
        seconds_remaining -= 1;
        timer_display.innerHTML = seconds_remaining;
        if (seconds_remaining < 1) {
        /**
        * The native JavaScript clearInterval() function stops
        * recurring function calls (such that the simulation object
        * represents a call to a particular function once per fixed
        * number of milliseconds).
        */
        clearInterval(simulation);
```

```
          finish_simulation();
          }
  }
  catch(e) {
          console.log("An exception to normal functioning occurred during the runtime of
decrement_timer_display_by_one_second(simulation): " + e);
  }
}

/**
 * Print a time-stamped message to the web browser console window
 * and to the web page element whose identifier is "seconds_elapsed_display" which
 * indicates the time at which the start() button was clicked.
 *
 * Hide the start() button to prevent overlapping timer processes.
 *
 * Set the span element whose identifier is "seconds_elapsed_display"
 * to display the nonnegative integer, N, which was most recently selected
 * inside of the select menu whose identifier is "nonnegative_integer_interval_selector".
 *
 * If no menu option was selected, set N to 30 by default.
 *
 * For N consecutive seconds, decrement the nonnegative integer value stored
 * in the span element whose identifier is "seconds_elapsed_display"
 * by exactly 1 until that value becomes 0.
 *
 * After the N-second timer interval completes, display the hidden start()
 * button so that it can be clicked again.
 */
function start() {
  try {
          // Display the time at which the START button was clicked on the web page and in the
browser console.
          let message = ("The start() button was clicked at time: " + generate_time_stamp()), N =
0, simulation = undefined;
          console.log(message);
          document.getElementById("application_status_messages_display").innerHTML += (('<'
+ 'p' + '>') + message + ('<' + '/' + 'p' + '>'));

          // Disable the start() button by hiding it until the simulation is finished.
          document.getElementById("timer_start_button").style.display = "none";

          N =
parseInt(get_selected_menu_option_value("nonnegative_integer_interval_selector"));
```

```
        // Ensure that N is an Number type datum, integer, and value no smaller than 30 and no
larger than 999.
        N = ((typeof N !== "number") || (N !== Math.floor(N)) || (N < 30) || (N > 999)) ? 30 : N;

        // Display the selected menu option on the web page and in the browser console.
        message = ("The selected value of N is " + N + ".");
        console.log(message);
        document.getElementById("application_status_messages_display").innerHTML += (('<'
+ 'p' + '>') + message + ('<' + '/' + 'p' + '>'));

        // Set the timer interval to N seconds and begin the incremental count down to 0 (in
increments 1 second in length).
        document.getElementById("seconds_elapsed_display").innerHTML = N;

        // Begin updating the timer display once per second (and 1 second is equal to 1000
milliseconds).
        simulation = setInterval( function() {
decrement_timer_display_by_one_second(simulation); }, 1000);
  }
  catch(e) {
        // Note: try-catch blocks are for "exception handling" (i.e. runtime error detection).
        console.log("An exception to normal functioning occurred during the runtime of start(): "
+ e);
  }
}
```

The following JavaScript (JS) code defines one function which controls the behavior of the COUNT_DOWN_TIMER web page application; namely the function which returns the number of milliseconds elapsed since the Unix Epoch concatenated to the string ” milliseconds since midnight on 01_JANUARY_1970″. Copy the JS code from the source code file which is linked below into a text editor and save that file as time_stamp.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/time_stamp.js

---

```
/**
 * file: time_stamp.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Get the Number of milliseconds which have elapsed since the Unix Epoch.
 * The Unix Epoch is 01_JANUARY_1970 at midnight (Coordinated Universal Time (UTC)).
 *
 * @return {String} message displaying the time at which this function was called.
 */
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.";
}
```

COUNT_DOWN_TIMER Interface (Initial)

The screenshot image below depicts what the COUNT_DOWN_TIMER web page interface is supposed to look like when the web page is initially loaded by a web browser.

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

---

COUNT_DOWN_TIMER Interface (Progress)

The screenshot image below depicts what the COUNT_DOWN_TIMER web page interface could look like after the end user selects a value for N from the expandable menu before clicking on the start() button to initiate a count-down timer interval lasting N consecutive seconds.

(Notice that the start() button is hidden while the count-down timer session is in progress).

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

---

COUNT_DOWN_TIMER Interface (Final)

The screenshot image below depicts what the COUNT_DOWN_TIMER web page interface could look like after N consecutive seconds elapse after the start() button is clicked.

(Notice that the start() button is unhidden after the count-down timer session finishes).

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

---

[End of abridged plain-text content from COUNT_DOWN_TIMER]

PI_APPROXIMATION

The single web page application featured in this tutorial web page substantiates a Monte Carlo dart-throwing simulation which plots a pixel-sized dot within a square canvas whose side length is 400 pixels once per second for a total of 3600 seconds (such that each dot is plotted at some random point whose two Cartesian plane coordinate values are each whole numbers). Dots which are plotted within a 200-pixel radius of the center of the canvas will be colored red. Dots which are plotted further than 200 pixels away from the center of the canvas will be colored blue. The quotient produced by dividing the total number of red dots by the sum of the total number of red dots and the total number of blue dots is the approximate value of Pi.

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

---

pi := (circle.circumference / circle.diameter). // true for all circles

pi_approximation := (4 * (red_pixel_count / (red_pixel_count + blue_pixel_count))).

Note that, like the Golden Ratio, Pi is an irrational number. Unlike each rational number, each irrational number cannot be represented as any integer divided by any nonzero integer.

A circle is a finite two-dimensional region such that each one of the points on the boundary of that region is the same distance away from exactly one center point (and that distance is a real number larger than zero).

The diameter of a circle is the length of the line segment which intersects the center of that circle and two unique points on the edge of that circle. The length of a circle's diameter is twice the length of that circle's radius.

A circle's radius is the length between any point on the edge of that circle and the point which is located at the center of that circle.

A circle's circumference is the length of the curved edge of that circle.

SOFTWARE_APPLICATION_COMPONENTS

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/pi_approximation.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_summer_2023_starter_pack/main/pi_approximation.js

---

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the PI_APPROXIMATION 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 pi_approximation.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/pi_approximation.html

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

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the PI_APPROXIMATION web page application. Copy the CSS code 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;
}
```

JavaScript Code

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

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/pi_approximation.js

---

```
/**
 * file: pi_approximation.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Generate a random x-value or a random y-value for a planar coordinate pair.
 *
 * @return {Number} an integer no smaller than -200 and no larger than 200.
 */
function generate_random_coordinate_scalar() {
  let random_nonnegative_integer_less_than_201 = Math.floor(Math.random() * 201);
  let random_number_sign = (Math.floor(Math.random() * 100) % 2 === 0) ? 1 : -1;
  return random_number_sign * random_nonnegative_integer_less_than_201;
}

/**
 * Generate a random coordinate pair for plotting points on a Cartesian plane whose origin is
represented by
 * the coordinate pair { x_coordinate: 0, y_coordinate: 0} and whose side lengths are no less
than 400 units.
 *
 * @return {Object} a planar point representation where the x-coordinate (i.e. horizontal position)
 *                  and the y-coordinate (i.e. vertical position) is each no smaller than -200 and no
larger than 200.
 */
function generate_random_planar_point() {
  return { x_coordinate : generate_random_coordinate_scalar(), y_coordinate:
generate_random_coordinate_scalar() };
}

/**
 * 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 < 0) throw "the function argument must be no smaller than zero.";
          n = input;
          a = n;
          b = 1;
          c = 0.000000001; // precision control
          while ((a - b) > 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;
  }
}

/**
 * Determine whether or not a given input value is a valid planar point object (as defined in the
generate_random_planar_point() function).
 *
 * @param {Object} input is assumed to be an object with the following properties:
 *        {Number} x_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
```

```
 *        {Number} y_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *
 * @return {Boolean} true if input satisfies the conditions defined above; false otherwise.
 */
function is_point(input) {
  try {
          if (arguments.length !== 1) throw "exactly one function argument is required.";
          if (typeof input.x_coordinate !== "number") throw "the x_coordinate property of input
must be a Number type value.";
          if (typeof input.y_coordinate !== "number") throw "the y_coordinate property of input
must be a Number type value.";
          if (Math.floor(input.x_coordinate) !== input.x_coordinate) throw "the x_coordinate
property of the input object must be a whole number value.";
          if (Math.floor(input.y_coordinate) !== input.y_coordinate) throw "the y_coordinate
property of the input object must be a whole number value.";
          if ((input.x_coordinate < -200) || (input.x_coordinate > 200)) throw "the x_coordinate
property of the input object must be no smaller than -200 and no larger than 200.";
          if ((input.y_coordinate < -200) || (input.y_coordinate > 200)) throw "the y_coordinate
property of the input object must be no smaller than -200 and no larger than 200.";
          return true;
  }
  catch(exception) {
          console.log("An exception to expected functioning occurred in is_point(input): " +
exception);
          return false;
  }
}

/**
 * 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 with the following properties:
 *        {Number} x_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *        {Number} y_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *
 * @param {Object} B is assumed to be an object with the following properties:
 *        {Number} x_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
```

```
 *        {Number} y_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *
 * @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 properties are as follows: {
x_coordinate : integer in range [-200,200], y_coordinate : integer in range [-200,200] }.";
          if (!is_point(B)) throw "B must be an object whose properties are as follows: {
x_coordinate : integer in range [-200,200], y_coordinate : integer in range [-200,200] }.";
          horizontal_difference = A.x_coordinate - B.x_coordinate;
          vertical_difference = A.y_coordinate - B.y_coordinate;
          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;
  }
}

/**
 * Add one to the number which is enclosed inside the HTML span element whose id is
"red_pixel_count_span" on the corresponding web page.
 *
 * Assume that the number of red pixels is a nonnnegative integer no larger than 3600.
 */
function increment_red_pixel_count() {
  let red_pixel_count_span = undefined, red_pixel_count = 0;
  try {
          red_pixel_count_span = document.getElementById("red_pixel_count_span");
          red_pixel_count = parseInt(red_pixel_count_span.innerHTML);
          if ((red_pixel_count < 0) || (red_pixel_count > 3600)) throw "red_pixel_count must be an
integer no smaller than 0 and no larger than 3600.";
          red_pixel_count += 1;
          red_pixel_count_span.innerHTML = red_pixel_count;
  }
  catch(exception) {
          console.log("An exception to expected functioning occurred in
increment_red_pixel_count(): " + exception);
```

```
  }
}

/**
 * Add one to the number which is enclosed inside the HTML span element whose id is
"blue_pixel_count_span" on the corresponding web page.
 *
 * Assume that the number of blue pixels is a nonnnegative integer no larger than 3600.
 */
function increment_blue_pixel_count() {
  let blue_pixel_count_span = undefined, blue_pixel_count = 0;
  try {
        blue_pixel_count_span = document.getElementById("blue_pixel_count_span");
        blue_pixel_count = parseInt(blue_pixel_count_span.innerHTML);
        if ((blue_pixel_count < 0) || (blue_pixel_count > 3600)) throw "blue_pixel_count must be
an integer no smaller than 0 and no larger than 3600.";
        blue_pixel_count += 1;
        blue_pixel_count_span.innerHTML = blue_pixel_count;
  }
  catch(exception) {
        console.log("An exception to expected functioning occurred in
increment_blue_pixel_count(): " + exception);
  }
}

/**
 * Retrieve the number which is enclosed inside the HTML span element whose id is
"red_pixel_count_span" on the corresponding web page.
 *
 * Assume that the number of red pixels is a nonnnegative integer no larger than 3600.
 *
 * @return {Number} an integer representing the total number of red pixel-sized "darts" which
have been plotted on the HTML canvas element on the corresponding web page.
 */
function get_red_pixel_count() {
  let red_pixel_count_span = undefined, red_pixel_count = 0;
  try {
        red_pixel_count_span = document.getElementById("red_pixel_count_span");
        red_pixel_count = parseInt(red_pixel_count_span.innerHTML);
        if ((red_pixel_count < 0) || (red_pixel_count > 3600)) throw "red_pixel_count must be an
integer no smaller than 0 and no larger than 3600.";
        return red_pixel_count;
  }
  catch(exception) {
```

```
		console.log("An exception to expected functioning occurred in get_red_pixel_count(): " +
exception);
		return 0;
  }
}

/**
 * Retrieve the number which is enclosed inside the HTML span element whose id is
"blue_pixel_count_span" on the corresponding web page.
 *
 * Assume that the number of blue pixels is a nonnnegative integer no larger than 3600.
 *
 * @return {Number} an integer representing the total number of blue pixel-sized "darts" which
have been plotted on the HTML canvas element on the corresponding web page.
 */
function get_blue_pixel_count() {
  let blue_pixel_count_span = undefined, blue_pixel_count = 0;
  try {
		blue_pixel_count_span = document.getElementById("blue_pixel_count_span");
		blue_pixel_count = parseInt(blue_pixel_count_span.innerHTML);
		if ((blue_pixel_count < 0) || (blue_pixel_count > 3600)) throw "blue_pixel_count must be
an integer no smaller than 0 and no larger than 3600.";
		return blue_pixel_count;
  }
  catch(exception) {
		console.log("An exception to expected functioning occurred in get_blue_pixel_count(): "
+ exception);
		return 0;
  }
}

/**
 * Set the number which is enclosed inside the HTML span element whose id is
"pi_approximation_span" to the result of the following computation:
 *
 * (4 * (red_pixel_count / (red_pixel_count + blue_pixel_count)))
 *
 * where red_pixel_count represents the total number of red pixel-sized "darts" which have been
plotted on the HTML canvas and
 * where blue_pixel_count represents the total number of blue pixel-sized "darts" which have
been plotted on the same HTML canvas
 * on the corresponding web page.
 *
 * Assume that this function is called once per second of the 3600 timed simulation.
 */
function update_pi_approximation() {
  let pi_approximation_span = undefined, pi_approximation = 0, red_pixel_count = 0,
blue_pixel_count = 0;
  try {
	red_pixel_count = get_red_pixel_count();
	blue_pixel_count = get_blue_pixel_count();
	pi_approximation_span = document.getElementById("pi_approximation_span");
	pi_approximation = (4 * (red_pixel_count / (red_pixel_count + blue_pixel_count)));
	pi_approximation_span.innerHTML = pi_approximation;
  }
  catch(exception) {
	console.log("An exception to expected functioning occurred in
update_pi_approximation(): " + exception);
  }
}

/**
 * Retrieve the number which is enclosed inside of the HTML span element whose id is
"seconds_remaining_span" on the corresponding web page.
 *
 * @return {Number} an integer which is assumed to be no smaller than 0 and no larger than
3600.
 */
function get_seconds_remaining() {
  let seconds_remaining_span = undefined, seconds_remaining = 0;
  try {
	seconds_remaining_span = document.getElementById("seconds_remaining_span");
	seconds_remaining = parseInt(seconds_remaining_span.innerHTML);
	if ((seconds_remaining < 0) || (seconds_remaining > 3600)) throw "seconds_remaining
must be an integer no smaller than 0 and no larger than 3600.";
	return seconds_remaining;
  }
  catch(exception) {
	console.log("An exception to expected functioning occurred in get_seconds_remaining():
" + exception);
  }
}

/**
 * Subtract one from the total number of seconds remaining (which is displayed inside the HTML
span element whose id is "seconds_remaining_span" on the corresponding web page).
 *
 * Assume that the number of seconds remaining is always an integer which is no smaller than 0
and no larger than 3600.
 */
function decrement_seconds_remaining() {
  let seconds_remaining_span = undefined, seconds_remaining = 0;
  try {
        seconds_remaining_span = document.getElementById("seconds_remaining_span");
        seconds_remaining = parseInt(seconds_remaining_span.innerHTML);
        if ((seconds_remaining < 0) || (seconds_remaining > 3600)) throw "seconds_remaining
must be an integer no smaller than 0 and no larger than 3600.";
        seconds_remaining -= 1;
        seconds_remaining_span.innerHTML = seconds_remaining;
  }
  catch(exception) {
        console.log("An exception to expected functioning occurred in
decrement_seconds_remaining(): " + exception);
        return 0;
  }
}

/**
 * Translate a coordinate pair from its Cartesian point representation to its HTML canvas
coordinate pair.
 *
 * The Cartesian point coordinates treat the center of the HTML canvas as the origin of the
Cartesian plane (i.e. where the x-coordinate is 0 and where the y-coordinate is 0).
 *
 * The HTML canvas (which is a square area whose side lengths are each 400 pixels) is
formatted such that the upper left corner has an x-coordinate of 0 and a y-coordinate of 0.
 *
 * @param {Object} point is assumed to be an object with the following properties:
 *      {Number} x_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *      {Number} y_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *
 * @return {Object} an object whose properties are as follows:
 *      {Number} x is assumed to be an integer no smaller than 0 and no larger than 400.
 *      {Number} y is assumed to be an integer no smaller than 0 and no larger than 400.
 */
function convert_point_to_pixel(point) {
  let pixel = { x : 0, y : 0 }; // initialize pixel to represent the top left corner of the HTML5 canvas.
  try {
        if (!is_point(point)) throw "point must be an object whose properties are as follows: {
x_coordinate : integer in range [-200,200], y_coordinate : integer in range [-200,200] }.";
        pixel.x = point.x_coordinate + 200;
        pixel.y = 200 - point.y_coordinate;
        return pixel;
  }
  catch(exception) {
        console.log("An exception to expected functioning occurred in
convert_point_to_pixel(point): " + exception);
  }
}

/**
 * Convert a Cartesian planar point to its corresponding HTML canvas point and plot it on the
400-by-400 pixel canvas element on the corresponding web page.
 *
 * If that pixel-sized "dart" is plotted within 200 pixels of the center of the square canvas, then
color the "dart" red.
 *
 * If that pixel-sized "dart" is plotted farther than 200 pixels away from the center of the square
canvas, then color the "dart" blue.
 *
 * @param {Object} point is assumed to be an object with the following properties:
 *      {Number} x_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 *      {Number} y_coordinate is assumed to be an integer no smaller than -200 and no larger
than 200.
 */
function plot_point_on_html_canvas(point) {
  let canvas = undefined, context = undefined, pixel = {}, distance_from_origin = 0;
  try {
        if (!is_point(point)) point = { x_coordinate : 0, y_coordinate : 0 };
        distance_from_origin = compute_distance_between_two_planar_points(point, {
x_coordinate : 0, y_coordinate : 0 });
        pixel = convert_point_to_pixel(point);
        canvas = document.getElementById("cartesian_plane");
        context =  canvas.getContext("2d");
        if (distance_from_origin > 200) { // outside of radius of circle inscribed inside of 400-pixel
square canvas
        context.beginPath();
        context.rect(pixel.x, pixel.y, 1, 1); // 1 pixel has a width of 1 and a height of 1
        context.strokeStyle = "#0000ff"; // HTML color code for blue.
        context.stroke();
        increment_blue_pixel_count();
        }
        else if (distance_from_origin <= 200) { // inside of radius of circle inscribed inside of
400-pixel square canvas
        context.beginPath();
        context.rect(pixel.x, pixel.y, 1, 1); // 1 pixel has a width of 1 and a height of 1
        context.strokeStyle = "#ff0000"; // HTML color code for red.
        context.stroke();
        increment_red_pixel_count();
        }
        else {
        throw "the pixel appears to be neither red nor blue.";
        }
        update_pi_approximation();

  }
  catch(exception) {
        console.log("An exception to expected functioning occurred in
plot_point_on_html_canvas(point): " + exception);
  }
}

/**
 * Draw a line whose thickness is one pixel and whose color is black from the middle of the left
edge of the HTML canvas to the middle of the right edge of that canvas.
 *
 * Assume that the canvas is 400 pixels in length on all sides.
 */
function draw_horizontal_line_through_middle_of_canvas() {
  let canvas = undefined, context = undefined;
  try {
        canvas = document.getElementById("cartesian_plane");
        context = canvas.getContext("2d");
        context.strokeStyle = "#000000";
        context.lineWidth = 1;
        context.beginPath();
        context.moveTo(0, 200); // middle point of left square canvas edge
        context.lineTo(400, 200); // middle point of right square canvas edge
        context.stroke();
  }
  catch(exception) {
        console.log("An exception to expected functioning occurred in
draw_horizontal_line_through_middle_of_canvas(): " + exception);
  }
}
```

```
/**
 * Draw a line whose thickness is one pixel and whose color is black from the middle of the top
edge of the HTML canvas to the middle of the bottom edge of that canvas.
 *
 * Assume that the canvas is 400 pixels in length on all sides.
 */
function draw_vertical_line_through_middle_of_canvas() {
	let canvas = undefined, context = undefined;
	try {
	canvas = document.getElementById("cartesian_plane");
	context = canvas.getContext("2d");
	context.strokeStyle = "#000000";
	context.lineWidth = 1;
	context.beginPath();
	context.moveTo(200, 0); // middle point of top square canvas edge
	context.lineTo(200, 400); // middle point of bottom square canvas edge
	context.stroke();
	}
	catch(exception) {
	console.log("An exception to expected functioning occurred in
draw_vertical_line_through_middle_of_canvas(): " + exception);
	}
}

/**
 * Remove any shapes which were plotted on the HTML canvas whose id is "cartesian_plane"
on the corresponding web page.
 *
 * Assume that this function is called in response to the RESET button being clicked.
 */
function clear_canvas() {
  let canvas = undefined, context = undefined;
  try {
	canvas = document.getElementById("cartesian_plane");
	context = canvas.getContext("2d");
	context.clearRect(0, 0, canvas.width, canvas.height);
	context.closePath();
  }
  catch(exception) {
	console.log("An exception to expected functioning occurred in clear_canvas(): " +
exception);
  }
}
```

```
/**
 * Make the START button invisible as soon as it is clicked and for the duration of the 3600
timed Monte Carlo dart-throwing simulation.
 *
 * Assume that this function is called in response to the START button being clicked.
 */
function hide_start_button() {
  try {
          let start_button = document.getElementById("start_button");
          start_button.style.display = "none";
  }
  catch(exception) {
          console.log("An exception to expected functioning occurred in hide_start_button(): " +
exception);
  }
}

/**
 * Make the START button visible as soon as it is clicked and for the duration of the 3600 timed
Monte Carlo dart-throwing simulation.
 *
 * Assume that this function is called in response to the RESET button being clicked.
 */
function unhide_start_button() {
  try {
          let start_button = document.getElementById("start_button");
          start_button.style.display = "block";
  }
  catch(exception) {
          console.log("An exception to expected functioning occurred in unhide_start_button(): " +
exception);
  }
}

/**
 * Make the RESET button invisible after the RESET button is clicked or after the web page is
loaded by the web browser.
 */
function hide_reset_button() {
  try {
          let reset_button = document.getElementById("reset_button");
          reset_button.style.display = "none";
  }
  catch(exception) {
          console.log("An exception to expected functioning occurred in hide_reset_button(): " +
exception);
  }
}

/**
 * Make the RESET button visible as soon as the 3600 timed Monte Carlo dart-throwing
simulation finishes.
 */
function unhide_reset_button() {
  try {
          let reset_button = document.getElementById("reset_button");
          reset_button.style.display = "block";
  }
  catch(exception) {
          console.log("An exception to expected functioning occurred in unhide_reset_button(): " +
exception);
  }
}

/**
 * Set the HTML canvas whose id is "cartesian_plane" to its initial state: no colored darts and
two perpendicular axis intersecting at the center of the canvas.
 *
 * Make the RESET button invisible (and therefore unclickable).
 *
 * Make the START button visible (and therefore clickable).
 *
 * Set the value enclosed by the span element whose id is "seconds_remaining_span" to 3600.
 *
 * Set the value enclosed by the span element whose id is "red_pixel_count_span" to 0.
 *
 * Set the value enclosed by the span element whose id is "blue_pixel_count_span" to 0.
 *
 * Set the value enclosed by the span element whose id is "pi_approximation" to 0.
 */
function initialize_application() {
  const time_point = Date.now(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
  let seconds_remaining_span, red_pixel_count_span, blue_pixel_count_span,
pi_approximation_span, timestamp, message, console_div;
  try {
          message = "The initialize_application() function was called at time: " + time_point + "
milliseconds since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).";
```

```
            console.log(message);
            hide_reset_button();
            unhide_start_button();
            clear_canvas();
            draw_horizontal_line_through_middle_of_canvas();
            draw_vertical_line_through_middle_of_canvas();
            seconds_remaining_span = document.getElementById("seconds_remaining_span");
            red_pixel_count_span = document.getElementById("red_pixel_count_span");
            blue_pixel_count_span = document.getElementById("blue_pixel_count_span");
            pi_approximation_span = document.getElementById("pi_approximation_span");
            console_div = document.getElementById("timestamped_events_log");
            seconds_remaining_span.innerHTML = 3600;
            red_pixel_count_span.innerHTML = 0;
            blue_pixel_count_span.innerHTML = 0;
            pi_approximation_span.innerHTML = 0;
            console_div.innerHTML += p0 + message + p1;
  }
  catch(exception) {
            console.log("An exception to expected functioning occurred in initialize_page(): " +
exception);
  }
}

/**
 * Print a time-stamped message to the web browser console which indicates that the timed
simulation has finished.
 *
 * Assume that this function is called in response to a timer interval session completing some
finite number of 3600 iterations
 * such that each iteration is temporally spaced by one second.
 *
 * Assume that plot_random_pixel_on_square_canvas(simulation) is the function which is called
3600 times
 * such that each plot_random_pixel_on_square_canvas(simulation) call is temporarlly spaced
by one second.
 */
function finish_simulation() {
  const time_point = Date.now(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
  const message = "The Monte Carlo simulation ended at time: "  + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).";
  console.log(message);
  document.getElementById("timestamped_events_log").innerHTML += p0 + message + p1;
}
```

```
/**
 * Call plot_random_pixel_on_square_canvas(simulation) once per second until
seconds_remaining is 0.
 *
 * Each time plot_random_pixel_on_square_canvas(simulation) is called, decrement
seconds_remaining by 1.
 *
 * @param {Object} simulation is assumed to be a timer interval handler (as defined in the
start_monte_carlo_simulation() function).
 */
function plot_random_pixel_on_square_canvas(simulation) {
  let point, seconds_remaining;
  point = generate_random_planar_point();
  decrement_seconds_remaining();
  seconds_remaining = get_seconds_remaining();
  plot_point_on_html_canvas(point);
  if (seconds_remaining === 0) {
            clearInterval(simulation);
            unhide_reset_button();
            finish_simulation();
  }
}

/**
 * Begin the 3600 second Monte Carlo dart throwing simulation.
 *
 * Assume that this function is called in response to the START button being clicked.
 *
 * Throw one random pixel-sized dart onto the square canvas per second for a total of 3600
seconds.
 *
 * Use the setInterval function to space plot_random_pixel_on_square_canvas(simulation)
function calls apart by one second.
 */
function start_monte_carlo_simulation() {
  const time_point = Date.now(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
  let simulation; // timer interval handler
  const message = "The Monte Carlo simulation started at time: " + time_point + " milliseconds
since 01_JANUARY_1970 00:00:00 (Coordinated Universal Time (UTC)).";
  console.log(message);
  document.getElementById("timestamped_events_log").innerHTML += p0 + message + p1;
  initialize_application();
  hide_start_button();
  simulation = setInterval( function() { plot_random_pixel_on_square_canvas(simulation); },
1000); // The plot_random_pixel_on_square_canvas(simulation) function is called once every
1000 milliseconds until the timer interval is cleared.
}
```

PI_APPROXIMATION Interface (Initial)

The screenshot image below depicts what the PI_APPROXIMATION web page interface is supposed to look like when the web page is initially loaded by a web browser or after the RESET button is clicked.

(Note that the RESET button is hidden until the 3600-second Monte Carlo dart-throwing simulation finishes).

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

---

PI_APPROXIMATION Interface (Progress 0)

The screenshot image below depicts what the PI_APPROXIMATION web page interface could look like 260 seconds after the START button is clicked.

(Notice that the START button is hidden while the 3600-second Monte Carlo dart-throwing simulation is in progress).

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/pi_approximation_interface_progress_0.png

---

PI_APPROXIMATION Interface (Progress 1)

The screenshot image below depicts what the PI_APPROXIMATION web page interface could look like 895 seconds after the START button is clicked.

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/pi_approximation_interface_progress_1.png

---

PI_APPROXIMATION Interface (Progress 2)

The screenshot image below depicts what the PI_APPROXIMATION web page interface could look like 2435 seconds after the START button is clicked.

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/pi_approximation_interface_progress_2.png

---

PI_APPROXIMATION Interface (Final)

The screenshot image below depicts what the PI_APPROXIMATION web page interface could look like 3600 seconds after the START button is clicked.

(Notice that the RESET button appears after the 3600-second Monte Carlo dart-throwing simulation finishes).

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

---

[End of abridged plain-text content from PI_APPROXIMATION]

BASE_CONVERTER

The single web page application featured in this tutorial web page converts an INPUT_SEQUENCE (i.e. an unsigned integer consisting of no more than eight digits) whose base is INPUT_BASE to its equivalent representation as an unsigned integer whose base is OUTPUT_BASE.

(The base conversion process used by this application is a more general version of the process used to convert unsigned integers from base-two to base-ten in the BITS_AND_BYTES tutorial web page of this website).

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

OUTPUT_SEQUENCE := CONVERT(INPUT_SEQUENCE, INPUT_BASE, OUTPUT_BASE). // First the base conversion process converts INPUT_SEQUENCE to its equivalent value in base-ten. Then that base-ten value is converted to its equivalent value in OUTPUT_BASE.

SOFTWARE_APPLICATION_COMPONENTS

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter.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_summer_2023_starter_packk/main/base_converter_interface.js

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter_arithmetic.js

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/time_stamp.js

---

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the BASE_CONVERTER 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 base_converter.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/base_converter.html

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

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the BASE_CONVERTER web page application. Copy the CSS code 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;
}
```

JavaScript Code

The following JavaScript (JS) code defines functions which control the behavior of the user interface component of the BASE_CONVERTER web page application. Copy the JS code from the source code file which is linked below into a text editor and save that file as base_converter_interface.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter_interface.js

---

```
/**
 * file: base_converter_interface.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Make a particular HTML element on the web page whose source code file is
base_converter.html invisible (without deleting that element).
 *
 * @param {String} element_id is assumed to be the id value of an existing HTML element on
the web whose source code file is base_converter.html.
 */
function hide_page_element(element_id) {
	try {
	let web_page_element;
	if (arguments.length !== 1) throw "exactly one function argument is required.";
	if (typeof element_id !== "string") throw "element_id is required to be a String type
value.";
	if (element_id.length < 1) throw "element_id is required to be a non-empty string.";
	web_page_element = document.getElementById(element_id);
	web_page_element.style.display = "none";
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of 
hide_page_element(element_id): " + exception);
	}
}

/**
 * Make a particular HTML element on the web page defined by base_converter.html visible 
rather than hidden.
 *
 * @param {String} element_id is assumed to be the id value of an existing HTML element on 
the web page whos source code file is base_converter.html.
 */
function unhide_page_element(element_id) {
	try {
	let web_page_element;
	if (arguments.length !== 1) throw "exactly one function argument is required.";
	if (typeof element_id !== "string") throw "element_id is required to be a String type 
value.";
	if (element_id.length < 1) throw "element_id is required to be a non-empty string.";
	web_page_element = document.getElementById(element_id);
	web_page_element.style.display = "block";
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of 
unhide_page_element(element_id): " + exception);
	}
}

/**
 * Generate the name of a particular numeric base in the following format: "[base_name]
(base-[base_number])".
 *
 * @param {Number} cardinality_of_base is assumed to be a base-ten integer which is no 
smaller than two and no larger than sixteen.
 *
 * @return {String} the inner HTML component of an option for a base select menu; undefined if 
an exception to normal functioning occurs.
 */
function get_numeric_base_option_label(cardinality_of_base) {
	try {
	if (arguments.length !== 1) throw "exactly one function argument is required.";
```

```
        if (typeof cardinality_of_base !== "number") throw "cardinality_of_base is required to be
a Number type value.";
        if (cardinality_of_base !== Math.floor(cardinality_of_base)) throw "cardinality_of_base is
required to be a whole number.";
        if ((cardinality_of_base < 2) || (cardinality_of_base > 16)) throw "cardinality_of_base is
required to be no smaller than two and no larger than sixteen.";
        if (cardinality_of_base === 2) return "BINARY (base-2)";
        if (cardinality_of_base === 3) return "TERNARY (base-3)";
        if (cardinality_of_base === 4) return "QUATERNARY (base-4)";
        if (cardinality_of_base === 5) return "QUINARY (base-5)";
        if (cardinality_of_base === 6) return "SENARY (base-6)";
        if (cardinality_of_base === 7) return "SEPTENARY (base-7)";
        if (cardinality_of_base === 8) return "OCTAL (base-8)";
        if (cardinality_of_base === 9) return "NONARY (base-9)";
        if (cardinality_of_base === 10) return "DECIMAL (base-10)";
        if (cardinality_of_base === 11) return "UNDECIMAL (base-11)";
        if (cardinality_of_base === 12) return "DUODECIMAL (base-12)";
        if (cardinality_of_base === 13) return "TRIDECIMAL (base-13)";
        if (cardinality_of_base === 14) return "TETRADECIMAL (base-14)";
        if (cardinality_of_base === 15) return "PENTADECIMAL (base-15)";
        if (cardinality_of_base === 16) return "HEXIDECIMAL (base-16)";
        if (true) throw "some unexpected runtime error occurred.";
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
get_numeric_base_option_label(cardinality_of_base): " + exception);
        return undefined;
        }
 }

/**
 * Define a select element whose options are the following natural number bases:
 * 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16.
 *
 * Note that the option whose value is "2" is set to be automatically selected if the application
user does not click on the select menu.
 *
 * @param {String} select_id is assumed to be the id of an existing select HTML element on the
corresponding web page.
 *
 * @return {String} an HTML string which is used to instantiate a select element whose options
are the following natural number bases:
 *              2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16;
 *              undefined if an exception to normal functioning occurs.
 */
function generate_html_string_for_base_select_menu(select_id) {
	try {
	let html_string = "", option_value = 0;
	if (arguments.length !== 1) throw "exactly one function argument is required.";
	if (typeof select_id !== "string") throw "select_id is required to be a String type value.";
	if (select_id.length < 1) throw "select_id is required to be a non-empty string.";
	html_string += '<' + 'select id="' + select_id + '"' + '>';
	html_string += '<' + 'option value="2" selected' + '>' + get_numeric_base_option_label(2)
+ '<' + '/' + 'option' + '>';
	for (option_value = 3; option_value <= 16; option_value += 1) html_string += '<' + 'option
value="' + option_value + '"' + '>' + get_numeric_base_option_label(option_value) + '<' + '/' +
'option' + '>';
	html_string += '<' + '/' + 'select' + '>';
	return html_string;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
generate_html_string_for_base_select_menu(select_id): " + exception);
	return undefined;
	}
 }

 /**
 * Return the value of the selected menu option of a select menu element.
 *
 * @param {String} select_menu_identifier is assumed to be the id of an existing select HTML
element on the corresponding web page.
 *
 * @return {String} the value of the selected menu option; undefined if an exception to normal
functioning occurs.
 */
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;
	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);
        return undefined;
        }
}

/**
 * Generate the digit key buttons which are used to enter digits whose base is
cardinality_of_base.
 *
 * For example, if cardinality_of_base is 16, then all sixteen digit key buttons will be generated.
 *
 * By contrast, if cardinality_of_base is 10, then only the first ten of sixteen digit key buttons will
be generated.
 *
 * The digit key buttons are arranged vertically in descending order (where each digit button is
encapsulated inside of its own paragraph element).
 *
 * @param {Number} cardinality_of_base is assumed to be a base-ten integer which is no
smaller than two and no larger than sixteen.
 *
 * @return {String} the HTML string which is used to substantiate the content of the div element
whose id is "input_sequence_digit_keys_div";
 *              undefined if an exception to normal functioning occurs.
 */
function generate_html_string_for_input_sequence_digit_buttons(cardinality_of_base) {
        try {
        let html_string = "", p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>', i = 0, hexidecimal_digit_set =
"0123456789ABCDEF";
        if (arguments.length !== 1) throw "exactly one function argument is required.";
        if (typeof cardinality_of_base !== "number") throw "cardinality_of_base is required to be
a Number type value.";
        if (cardinality_of_base !== Math.floor(cardinality_of_base)) throw "cardinality_of_base is
required to be a whole number.";
        if ((cardinality_of_base < 2) || (cardinality_of_base > 16)) throw "cardinality_of_base is
required to be no smaller than two and no larger than sixteen.";
        for (i = 0; i < cardinality_of_base; i += 1) html_string += p0 + '<' + 'input type="button"
value="' + hexidecimal_digit_set[i] + '" onclick="append_digit_' + hexidecimal_digit_set[i] + '()"' +
'>' + p1;
        return html_string;
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
generate_html_string_for_input_sequence_digit_buttons(cardinality_of_base): " + exception);
```

```
		return undefined;
		}
}

/**
 * Determine whether or not hexidecimal_digit is an element of the following set:
 * ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' ].
 *
 * @param {String} hexidecimal_digit is assumed to be an element of the following set:
 *			['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' ].
 *
 * @return {Boolean} true if hexidecimal is a valid hexidecimal digit as defined above; false
otherwise.
 */
function is_valid_hexidecimal_digit(hexidecimal_digit) {
		try {
		let i = 0, hexidecimal_digit_set = "0123456789ABCDEF";
		if (arguments.length !== 1) throw "exactly one function argument is required.";
		if (typeof hexidecimal_digit !== "string") throw "hexidecimal_digit is required to be a
String type value.";
		if (hexidecimal_digit.length !== 1) throw "hexidecimal_digit is required to have a length of
exactly one character.";
		for (i = 0; i < 16; i += 1) if (hexidecimal_digit === hexidecimal_digit_set[i]) return true;
		return false;
		}
		catch(exception) {
		console.log("An exception to normal functioning occurred during the runtime of
is_valid_hexidecimal_digit(hexidecimal_digit): " + exception);
		return false;
		}
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append hexidecimal_digit to right end of that string (such that the total number of characters
enclosed by the input sequence span is always eight).
 *
 * @param {String} hexidecimal_digit is assumed to be an element of the following set:
 *			['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' ].
 */
function append_digit(hexidecimal_digit) {
		try {
```

```
	let i = 0, input_sequence_span, old_input_sequence = "", new_input_sequence = "";
	if (!is_valid_hexidecimal_digit(hexidecimal_digit)) throw "hexidecimal_digit is required to
be an element of the following set: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' ].";
	input_sequence_span = document.getElementById("input_sequence_span");
	old_input_sequence = input_sequence_span.innerHTML;
	for (i = 1; i < 8; i += 1) new_input_sequence += old_input_sequence[i];
	new_input_sequence += hexidecimal_digit;
	input_sequence_span.innerHTML = new_input_sequence;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
append_digit_1(): " + exception);
	}
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '0' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_0() {
	append_digit('0');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '1' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_1() {
	append_digit('1');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '2' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
```

```
function append_digit_2() {
	append_digit('2');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '3' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_3() {
	append_digit('3');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '4' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_4() {
	append_digit('4');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '5' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_5() {
	append_digit('5');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '6' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
```

```
function append_digit_6() {
	append_digit('6');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '7' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_7() {
	append_digit('7');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '8' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_8() {
	append_digit('8');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append '9' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_9() {
	append_digit('9');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append 'A' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
```

```
function append_digit_A() {
	append_digit('A');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append 'B' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_B() {
	append_digit('B');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append 'C' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_C() {
	append_digit('C');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append 'D' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_D() {
	append_digit('D');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append 'E' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
```

```
function append_digit_E() {
        append_digit('E');
}

/**
 * Remove the leftmost character from the inner HTML string of the span HTML element whose
id is "input_sequence_span".
 *
 * Append 'F' to right end of that string (such that the total number of characters enclosed by the
input sequence span is always eight).
 */
function append_digit_F() {
        append_digit('F');
}

/**
 * Set the content displayed inside of the span HTML element whose id is
"input_sequence_span" to "00000000".
 */
function initialize_input_sequence_span() {
        try {
        document.getElementById("input_sequence_span").innerHTML = "00000000";
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
initialize_input_sequence_span(): " + exception);
        }
}

/**
 * Set the inner HTML of the span element whose id is "input_sequence_span_2" and
 * "input_sequence_span_3" to the inner HTML of the span element whose id is
"input_sequence_span".
 */
function update_input_sequence_span() {
        try {
        let input_sequence_span, input_sequence_span_2, input_sequence_span_3;
        input_sequence_span = document.getElementById("input_sequence_span");
        input_sequence_span_2 = document.getElementById("input_sequence_span_2");
        input_sequence_span_3 = document.getElementById("input_sequence_span_3");
        input_sequence_span_2.innerHTML = input_sequence_span.innerHTML;
        input_sequence_span_3.innerHTML = input_sequence_span.innerHTML;
        }
        catch(exception) {
          console.log("An exception to normal functioning occurred during the runtime of
update_input_sequence_span_2(): " + exception);
          }
}

/**
 * Set the content displayed inside of
 * the span element whose id is "input_base_span",
 * the span element whose id is "input_base_span_2", and
 * the span element whose id is "input_base_span_3" to the
 * input base option which is currently selected from the list of options
 * displayed inside the select HTML element whose id is "input_base_menu".
 */
function update_input_base_span() {
          try {
          let input_base_value = 0, input_base_span, input_base_span_2, input_base_span_3;
          input_base_span = document.getElementById("input_base_span");
          input_base_span_2 = document.getElementById("input_base_span_2");
          input_base_span_3 = document.getElementById("input_base_span_3");
          input_base_value = get_selected_menu_option_value("input_base_menu");
          input_base_span.innerHTML = input_base_value;
          input_base_span_2.innerHTML = input_base_value;
          input_base_span_3.innerHTML = input_base_value;
          }
          catch(exception) {
          console.log("An exception to normal functioning occurred during the runtime of
update_input_base_span(): " + exception);
          }
}

/**
 * Set the content displayed inside of the HTML element whose id is "digit_buttons_div"
 * to display the digit buttons which correspond with the input base option
 * which is currently selected from the list of options displayed inside the select HTML element
whose id is "input_base_menu".
 */
function update_digit_buttons_div() {
          try {
          let input_base_value = 0, digit_buttons_div, digit_buttons_for_selected_base;
          digit_buttons_div = document.getElementById("digit_buttons_div");
          input_base_value = parseInt(get_selected_menu_option_value("input_base_menu"));
          digit_buttons_for_selected_base =
generate_html_string_for_input_sequence_digit_buttons(input_base_value);
          digit_buttons_div.innerHTML = digit_buttons_for_selected_base;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of update_input_base_span(): " + exception);
	}
}

/**
 * Set the content displayed inside of the span element whose id is "output_base_span_2"
 * to the output base option which is currently selected from the list of options displayed
 * inside the select HTML element whose id is "output_base_menu".
 */
function update_output_base_span_2() {
	try {
	let output_base_value = 0, output_base_span_2;
	output_base_span_2 = document.getElementById("output_base_span_2");
	output_base_value = get_selected_menu_option_value("output_base_menu");
	output_base_span_2.innerHTML = output_base_value;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of update_output_base_span_2(): " + exception);
	}
}

/**
 * Progress from STEP_0 to STEP_1 of the base converter application's input form (which is substantiated using the web page file named base_converter.html).
 *
 * Assume that this function is called in response to the event of a button click of the NEXT button displayed inside of the div whose id is "step_0_div".
 *
 * Submit the input base option which is currently selected from the list of options displayed inside the select HTML element whose id is "input_base_menu".
 *
 * Display the digit buttons which correspond with the selected input base in the next step.
 *
 * Append a time stamped message to the bottom of the web page indicating that this function was called.
 */
function step_0_next() {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "step_0_next() was called at time: " + time_stamp;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
	update_input_base_span();
	update_digit_buttons_div();
	hide_page_element("step_0_div");
	unhide_page_element("step_1_div");
}

/**
 * Regress from STEP_1 to STEP_0 of the base converter application's input form (which is
substantiated using the web page file named base_converter.html).
 *
 * Assume that this function is called in response to the event of a button click of the BACK
button displayed inside of the div whose id is "step_1_div".
 *
 * Reset the text displayed inside of the span element whose id is "input_sequence_span" to its
initial state: 00000000.
 *
 * Append a time stamped message to the bottom of the web page indicating that this function
was called.
 */
function step_1_back() {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "step_1_back() was called at time: " + time_stamp;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
	initialize_input_sequence_span();
	hide_page_element("step_1_div");
	unhide_page_element("step_0_div");
}

/**
 * Progress from STEP_1 to STEP_2 of the base converter application's input form (which is
substantiated using the web page file named base_converter.html).
 *
 * Assume that this function is called in response to the event of a button click of the NEXT
button displayed inside of the div whose id is "step_1_div".
 *
 * Update the input base and input sequence displays inside of the STEP_2 div.
 *
 * Append a time stamped message to the bottom of the web page indicating that this function
was called.
 */
```

```
function step_1_next() {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "step_1_next() was called at time: " + time_stamp;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
	update_input_sequence_span();
	hide_page_element("step_1_div");
	unhide_page_element("step_2_div");
}

/**
 * Regress from STEP_2 to STEP_1 of the base converter application's input form (which is
substantiated using the web page file named base_converter.html).
 *
 * Assume that this function is called in response to the event of a button click of the BACK
button displayed inside of the div whose id is "step_2_div".
 *
 * Append a time stamped message to the bottom of the web page indicating that this function
was called.
 */
function step_2_back() {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "step_2_back() was called at time: " + time_stamp;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
	initialize_input_sequence_span();
	hide_page_element("step_2_div");
	unhide_page_element("step_1_div");
}

/**
 * Progress from STEP_2 to STEP_3 of the base converter application's input form (which is
substantiated using the web page file named base_converter.html).
 *
 * Assume that this function is called in response to the event of a button click of the NEXT
button displayed inside of the div whose id is "step_2_div".
 *
 * Submit the output base option which is currently selected from the list of options displayed
inside the select HTML element whose id is "input_base_menu".
 *
 * Update the input_base, input_sequence, and output_base values which are displayed inside
the STEP_3 div.
```

```
 *
 * Append a time stamped message to the bottom of the web page indicating that this function
was called.
 */
function step_2_next() {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "step_2_next() was called at time: " + time_stamp;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
	update_output_base_span_2();
	hide_page_element("step_2_div");
	unhide_page_element("step_3_div");
}

/**
 * Regress from STEP_3 to STEP_2 of the base converter application's input form (which is
substantiated using the web page file named base_converter.html).
 *
 * Assume that this function is called in response to the event of a button click of the BACK
button displayed inside of the div whose id is "step_3_div".
 *
 * Reset the text displayed inside of the div element whose id is "output" to its initial state: "This
sentence will be replaced with base conversion results when the CONVERT button is clicked."
 *
 * Append a time stamped message to the bottom of the web page indicating that this function
was called.
 */
function step_3_back() {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "step_3_back() was called at time: " + time_stamp;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
	initialize_output_div();
	hide_page_element("step_3_div");
	unhide_page_element("step_2_div");
}

/**
 * Define the application interface component (to be displayed on the web page whose source
code file is base_converter.html)
 * which enables the user to select an INPUT_BASE.
 *
 * @return {String} an HTML string which is used to instantiate a DIV page element whose id is
"step_0_div".
 */
function generate_html_string_for_step_0_div() {
	let html_string = '<' + 'div class="module input_step" id="step_0_div"' + '>', p0 = '<' + 'p' +
'>', p1 ='<' + '/' + 'p' + '>';
	html_string += p0 + "STEP_0: Select an INPUT_BASE from the expandable menu. Then
click the NEXT button to proceed to the INPUT_SEQUENCE step." + p1;
	html_string += p0 + "INPUT_BASE := " +
generate_html_string_for_base_select_menu("input_base_menu") + p1;
	html_string += p0 + '<' + 'input type="button" value="NEXT" onclick="step_0_next()"' +
'>';
	html_string += '<' + '/' + 'div' + '>';
	return html_string;
}

/**
 * Define the application interface component (to be displayed on the web page whose source
code file is base_converter.html)
 * which enables the user to enter an INPUT_SEQUENCE.
 *
 * @return {String} an HTML string which is used to instantiate a DIV page element whose id is
"step_1_div".
 */
function generate_html_string_for_step_1_div() {
	let html_string = '<' + 'div class="module input_step" id="step_1_div"' + '>', p0 = '<' + 'p' +
'>', p1 = '<' + '/' + 'p' + '>', input_sequence_span;
	input_base_value = get_selected_menu_option_value("input_base_menu");
	html_string += p0 + "STEP_1: Use the digit keys to enter no more than eight digits. Then
click the NEXT button to proceed to the INPUT_SEQUENCE step. To re-select the
INPUT_BASE, click the BACK button." + p1;
	html_string += p0 + "INPUT_BASE := " + '<' + 'span
style="background:#000000;color:#ffff00" id="input_base_span"' + '>' + input_base_value + '<' +
'/' + 'span' + '>' + p1;
	html_string += p0 + "INPUT_SEQUENCE := " + '<' + 'span
style="background:#000000;color:#ffff00" id="input_sequence_span"' + '>' + "00000000" + '<' +
'/' + 'span' + '>' + p1;
	html_string += '<' + 'div id="digit_buttons_div"' + '>' +
generate_html_string_for_input_sequence_digit_buttons(parseInt(input_base_value)) + '<' + '/' +
'div' + '>';
	html_string += p0 + '<' + 'input type="button" value="BACK" onclick="step_1_back()"' +
'>';
	html_string += " " + '<' + 'input type="button" value="NEXT" onclick="step_1_next()"' + '>'
+ p1;
```

```
	html_string += '<' + '/' + 'div' + '>';
	return html_string;
}

/**
 * Define the application interface component (to be displayed on the web page whose source
code file is base_converter.html)
 * which enables the user to review input values before initiating the base conversion.
 *
 * @return {String} an HTML string which is used to instantiate a DIV page element whose id is
"step_2_div".
 */
function generate_html_string_for_step_2_div() {
	let html_string = '<' + 'div class="module input_step" id="step_2_div"' + '>', p0 = '<' + 'p' +
'>', p1 ='<' + '/' + 'p' + '>';
	html_string += p0 + "STEP_2: Select an OUTPUT_BASE from the expandable menu.
Then click the NEXT button to proceed to the BASE_CONVERSION step. To re-enter the
INPUT_SEQUENCE, click the BACK button." + p1;
	html_string += p0 + "INPUT_BASE := " + '<' + 'span
style="background:#000000;color:#ffff00" id="input_base_span_2"' + '>' + "???" + '<' + '/' +
'span' + '>' + p1;
	html_string += p0 + "INPUT_SEQUENCE := " + '<' + 'span
style="background:#000000;color:#ffff00" id="input_sequence_span_2"' + '>' + "???" + '<' + '/' +
'span' + '>' + p1;
	html_string += p0 + "OUTPUT_BASE := " + '<' + 'span
style="background:#000000;color:#ffff00" id="output_base_span"' + '>' +
generate_html_string_for_base_select_menu("output_base_menu") + '<' + '/' + 'span' + '>' + p1;
	html_string += p0 + '<' + 'input type="button" value="BACK" onclick="step_2_back()"' +
'>';
	html_string += " " + '<' + 'input type="button" value="NEXT" onclick="step_2_next()"' + '>'
+ p1;
	html_string += '<' + '/' + 'div' + '>';
	return html_string;
}

/**
 * Define the application interface component (to be displayed on the web page whose source
code file is base_converter.html)
 * which enables the user to click the CONVERT button.
 *
 * @return {String} an HTML string which is used to instantiate a DIV page element whose id is
"step_3_div".
 */
function generate_html_string_for_step_3_div() {
	let html_string = '<' + 'div class="module input_step" id="step_3_div"' + '>', p0 = '<' + 'p' + 
'>', p1 ='<' + '/' + 'p' + '>';
	html_string += p0 + "STEP_3: Click the CONVERT button to convert 
INPUT_SEQUENCE to its OUTPUT_BASE representation. To re-select the OUTPUT_BASE, 
click the BACK button." + p1;
	html_string += p0 + "INPUT_BASE := " + '<' + 'span 
style="background:#000000;color:#ffff00" id="input_base_span_3"' + '>' + "???" + '<' + '/' + 
'span' + '>' + p1;
	html_string += p0 + "INPUT_SEQUENCE := " + '<' + 'span 
style="background:#000000;color:#ffff00" id="input_sequence_span_3"' + '>' + "???" + '<' + '/' + 
'span' + '>' + p1;
	html_string += p0 + "OUTPUT_BASE := " + '<' + 'span 
style="background:#000000;color:#ffff00" id="output_base_span_2"' + '>' + "???" + '<' + '/' + 
'span' + '>' + p1;
	html_string += p0 + '<' + 'input type="button" value="BACK" onclick="step_3_back()"' + 
'>';
	html_string += " " + '<' + 'input type="button" value="CONVERT" onclick="convert()"' + '>' 
+ p1;
	html_string += '<' + '/' + 'div' + '>';
	return html_string;
}

/**
 * Set the content displayed inside of the div HTML element whose id is "output" to its initial 
placeholder text.
 */
function initialize_output_div() {
	try {
	let p0 = '<' + 'p class="module output_step"' + '>', p1 = '<' + '/' + 'p' + '>';
	document.getElementById("output").innerHTML = p0 + "This sentence will be replaced 
with base conversion results when the CONVERT button is clicked." + p1;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of 
initialize_output_div(): " + exception);
	}
}

/**
 * Set the web page interface (i.e. base_converter.html) to its intial state.
 *
 * Assume that this function is called in response the event of the web page being loaed by the 
web browser.
 */
```

```
function initialize_application() {
	try {
	const time_stamp = generate_time_stamp(), p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let message = "The web page, base_converter.html, was loaded at time: " + time_stamp;
	let input_form_div = undefined;
	console.log(message);
	document.getElementById("time_stamped_messages").innerHTML = p0 + message +
p1;
	input_form_div = document.getElementById("input_form");
	initialize_output_div();
	input_form_div.innerHTML = generate_html_string_for_step_0_div();
	input_form_div.innerHTML += generate_html_string_for_step_1_div();
	input_form_div.innerHTML += generate_html_string_for_step_2_div();
	input_form_div.innerHTML += generate_html_string_for_step_3_div();
	hide_page_element("step_1_div");
	hide_page_element("step_2_div");
	hide_page_element("step_3_div");
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
initialize_application(): " + exception);
	}
}

/**
 * Perform the base converion using the input values which were submitted by the application
user via the web page interface.
 *
 * Convert an INPUT_SEQUENCE (i.e. an unsigned integer consisting of no more than eight
digits)
 * whose base is INPUT_BASE to its equivalent representation as an unsigned integer whose
base is OUTPUT_BASE.
 *
 * Append a time stamped message to the bottom of the web page indicating that this function
was called.
 */
function convert() {
	try {
	const time_stamp = generate_time_stamp();
	let message = "convert() was called at time: " + time_stamp;
	let INPUT_BASE = 0, INPUT_SEQUENCE = "", OUTPUT_BASE = 0,
OUTPUT_SEQUENCE = "";
	let  p00 = '<' + 'p class="module" style="border-color: #00ffff; border-width:
1px;border-style: solid;"' + '>', p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
```

```
        console.log(message);
        document.getElementById("time_stamped_messages").innerHTML += p0 + message +
p1;
        INPUT_BASE = parseInt(document.getElementById("input_base_span").innerHTML);
        INPUT_SEQUENCE = document.getElementById("input_sequence_span").innerHTML;
        OUTPUT_BASE =
parseInt(document.getElementById("output_base_span_2").innerHTML);
        OUTPUT_SEQUENCE = perform_base_conversion(INPUT_BASE,
INPUT_SEQUENCE, OUTPUT_BASE);
        document.getElementById("output").innerHTML = p00 + INPUT_SEQUENCE + " (in
base-" + INPUT_BASE + ") is equal to " + OUTPUT_SEQUENCE + " (in base-" +
OUTPUT_BASE + ")." + p1
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of convert():
" + exception);
        }
}
```

JavaScript Code

The following JavaScript (JS) code defines functions which perform the arithmetic operations involved in converting INPUT_SEQUENCE from INPUT_BASE to OUTPUT_BASE when using the BASE_CONVERTER web page application. Copy the JS code from the source code file which is linked below into a text editor and save that file as base_converter_arithmetic.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter_arithmetic.js

---

```
/**
 * file: base_converter_arithmetic.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
```

```
 * Get the result of the following calculation: base ^ exponent (i.e. base to the power of exponent
(i.e. base multiplied by itself exponent times))
 * .
 * @param {Number} base is assumed to be a base-ten natural number which is no smaller than
two and no larger than sixteen.
 *
 * @param {Number} exponent is assumed to be a base-ten natural number which is no smaller
than zero and no larger than seven.
 *
 * @return {Number} base multiplied by itself exponent times if no exception is thrown;
otherwise 1.
 */
function compute_power(base, exponent) {
    try {
        let result = 1;
        if (arguments.length !== 2) throw "exactly two function arguments are required.";
        if (typeof base !== "number") throw "base is required to be a Number type value.";
        if (typeof exponent !== "number") throw "exponent is required to be a Number type value.";
        if (Math.floor(base) !== base) throw "base is required to be a whole number.";
        if (Math.floor(exponent) !== exponent) throw "exponent is required to be a whole number.";
        if ((base < 2) || (base > 16)) throw "base is required to be no smaller than two and no larger
than sixteen.";
        if ((exponent < 0) || (exponent > 7)) throw "exponent is required to be no smaller than zero
and no larger than seven.";
        while (exponent > 0) {
            result *= base;
            exponent -= 1;
        }
        return result;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
compute_power(base, exponent): " + exception);
        return 1;
    }
}

/**
 * Determine whether or not the given function arguments represents a valid input_sequence
with whose digits are in the input_base.
 *
 * @param {Number} input_base is assumed to be a base-ten integer which is no smaller than
two and no larger than sixteen.
 *
 * @param {String} input_sequence is assumed to be a sequence of exactly eight characters
such that each character is a digit whose numeric base is input_base.
 *
 * @return {Boolean} true if input_sequence is a string comprised of exactly eight characters
such that each character is a digit whose numeric base is input_base; false otherwise.
 */
function validate_input_sequence(input_base, input_sequence) {
    try {
        let hexidecimal_digit_set = "0123456789ABCDEF", i = 0, k = 0,
is_valid_input_sequence_digit = false;
        if (arguments.length !== 2) throw "exactly two function arguments are required.";
        if (typeof input_base !== "number") throw "input_base is required to be a Number type
value.";
        if (typeof input_sequence !== "string") throw "input_sequence is required to be a String
type value.";
        if (Math.floor(input_base) !== input_base) throw "input_base is required to be a whole
number.";
        if ((input_base < 2) || (input_base > 16)) throw "input_base is required to be no smaller
than two and no larger than sixteen.";
        if (input_sequence.length !== 8) throw "input_sequence is required to have a length of
exactly eight characters.";
        for (i = 0; i < 8; i++) {
            for (k = 0; k < input_base; k++) if (input_sequence[i] === hexidecimal_digit_set[k])
is_valid_input_sequence_digit = true;
            if (!is_valid_input_sequence_digit) throw "at least one character in input_sequence is not
a valid digit (i.e. a digit whose base is input_base).";
            is_valid_input_sequence_digit = false;
        }
        return true;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_to_decimal(input_base, input_sequence): " + exception);
        return false;
    }
}

/**
 * Return the decimal integer index of the hexidecimal digit which H represents where
hexidecimal digits
 * are the following set of characters arranged in the following permutation:
 * ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']
 *
 * @param {String} H is assumed to be a single hexidecimal digit.
 *
 * @return {Number} the decimal integer index of the hexidecimal digit which H represents.
 */
function get_decimal_value_of_hexidecimal_digit(H) {
    if (H === '0') return 0;
    if (H === '1') return 1;
    if (H === '2') return 2;
    if (H === '3') return 3;
    if (H === '4') return 4;
    if (H === '5') return 5;
    if (H === '6') return 6;
    if (H === '7') return 7;
    if (H === '8') return 8;
    if (H === '9') return 9;
    if (H === 'A') return 10;
    if (H === 'B') return 11;
    if (H === 'C') return 12;
    if (H === 'D') return 13;
    if (H === 'E') return 14;
    if (H === 'F') return 15;
    return 0;
}

/**
 * Get the decimal number representation of input_sequence by converting input_sequence
from input_base to decimal (i.e. base-ten).
 *
 * @param {Number} input_base is assumed to be a base-ten integer which is no smaller than
two and no larger than sixteen.
 *
 * @param {String} input_sequence is assumed to be a sequence of exactly eight characters
such that each character is a digit whose numeric base is input_base.
 *
 * @return {Number} the base-ten number which input_sequence in the input_base represents;
zero if an exception to normal functioning is thrown.
 */
function convert_to_decimal(input_base, input_sequence) {
    try {
        let decimal_output = 0, i = 0;
        if (!validate_input_sequence(input_base, input_sequence)) throw "input_base or
input_sequence appears to be an invalid function argument.";
        for (i = 0; i < 8; i++) decimal_output +=
(get_decimal_value_of_hexidecimal_digit(input_sequence[i]) * compute_power(input_base, 7 -
i));
```

```
        return decimal_output;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_to_decimal(input_base, input_sequence): " + exception);
        return 0;
    }
}

/**
 * Get the hexidecimal equivalent of a single decimal digit.
 *
 * @param {Number} D is assumed to be a single base-ten digit (i.e. exactly one element within
the following array:
 *              [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).
 *
 * @return {String} the element of the following array whose index in the array is D:
 *              ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'] if D is a valid decimal digit;
 *              '0' otherwise.
 */
function convert_decimal_digit_to_hexidecimal(D) {
    try {
        let hexidecimal_digit_set = "0123456789ABCDEF";
        if (arguments.length !== 1) throw "exactly one function argument is required.";
        if (typeof D !== "number") throw "D is required to be a Number type value.";
        if (Math.floor(D) !== D) throw "D is required to be a whole number.";
        if ((D < 0) || (D > 9)) "D is required to be no smaller than zero and no larger than nine.";
        return hexidecimal_digit_set[D];
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_decimal_digit_to_hexidecimal(D): " + exception);
        return '0';
    }
}

/**
 * Return a string whose constituent characters are identical to the characters in S except for the
fact that
 * the output string characters are in reverse order from those in S.
 *
 * @param {String} S is assumed to be a sequence of characters.
 *
 * @return {String} S in reverse order.
 */
function reverse_string(S) {
    try {
        let output_string = "", i = 0;
        if (arguments.length !== 1) throw "exactly one function argument is required.";
        if (typeof S !== "string") throw "output_string is required to be a String type value.";
        for (i = (S.length - 1); i >= 0; i--) output_string += S[i];
        return output_string;
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of 
reverse_string(S): " + exception);
        return "ERROR";
    }
}

/**
 * Get the output_base representation of decimal_number by converting decimal_number from 
decimal (i.e. base-ten) to output_base.
 *
 * @param {Number} output_base is assumed to be a base-ten integer which is no smaller than 
two and no larger than sixteen.
 *
 * @param {Number} decimal_number is assumed to be a base-ten integer no smaller than zero 
and no larger than the following sum:
 *                (16 * (16 ^ 7)) + (16 * (16 ^ 6)) + (16 * (16 ^ 5)) + (16 * (16 ^ 4)) + (16 * (16 ^ 3)) + 
(16 * (16 ^ 2)) + (16 * (16 ^ 1)) + (16 * (16 ^ 0)) = 4581298448.
 *
 * @return {String} a sequence of one ore more hexidecimal digits which represents the value of 
decimal_number in the given output_base.
 */
function convert_from_decimal(output_base, decimal_number) {
    try {
        let maximum_decimal_number_value = 0, i = 0, output_base_result = "";
        if (arguments.length !== 2) throw "exactly two function arguments are required.";
        if (typeof output_base !== "number") throw "output_base is required to be a Number type 
value.";
        if (typeof decimal_number !== "number") throw "decimal_number is required to be a 
Number type value.";
        if (Math.floor(output_base) !== output_base) throw "output_base is required to be a whole 
number.";
        if (Math.floor(decimal_number) !== decimal_number) throw "decimal_number is required to 
be a whole number.";
```

```
        if ((output_base < 2) || (output_base > 16)) throw "output_base is required to be no smaller
than two and no larger than sixteen.";
        for (i = 7; i >= 0; i -= 1) maximum_decimal_number_value += (16 * compute_power(16, i));
        if ((decimal_number < 0) || (decimal_number > maximum_decimal_number_value)) throw
"decimal_number is required to be larger than zero and no larger than " +
maximum_decimal_number_value + ".";
        while (decimal_number > 0) {
            output_base_result += convert_decimal_digit_to_hexidecimal(decimal_number %
output_base);
            decimal_number = Math.floor(decimal_number / output_base);
        }
        return reverse_string(output_base_result);
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
convert_from_decimal(output_base, decimal_number): " + exception);
        return '0';
    }
}

/**
 * Convert an INPUT_SEQUENCE (i.e. an unsigned integer consisting of no more than eight
digits)
 * whose base is INPUT_BASE to its equivalent representation as an unsigned integer whose
base is OUTPUT_BASE.
 *
 * This is the "main function" of this JavaScript file (i.e. the "macro function" which implements
each of the arithmetic functions defined above).
 *
 * @param {Number} input_base is assumed to be a base-ten integer which is no smaller than
two and no larger than sixteen.
 *
 * @param {String} input_sequence is assumed to be a sequence of exactly eight characters
such that each character is a digit whose numeric base is input_base.
 *
 * @param {Number} output_base is assumed to be a base-ten integer which is no smaller than
two and no larger than sixteen.
 *
 * @return {String} input_sequence in its output_base representation if no exception to normal
functioning is thrown; "0" otherwise.
 */
function perform_base_conversion(input_base, input_sequence, output_base) {
    try {
        let decimal_number = convert_to_decimal(input_base, input_sequence);
        return convert_from_decimal(output_base, decimal_number);
    }
    catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
perform_base_conversion(input_base, input_sequence, output_base): " + exception);
        return '0';
    }
}
```

JavaScript Code

The following JavaScript (JS) code defines one function which controls the behavior of the HEXIDECIMAL_COLOR_CODES web page application; namely the function which returns the number of milliseconds elapsed since the Unix Epoch concatenated to the string ” milliseconds since midnight on 01_JANUARY_1970″. Copy the JS code from the source code file which is linked below into a text editor and save that file as time_stamp.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/time_stamp.js

---

```
/**
 * file: time_stamp.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Get the Number of milliseconds which have elapsed since the Unix Epoch.
 * The Unix Epoch is 01_JANUARY_1970 at midnight (Coordinated Universal Time (UTC)).
 *
 * @return {String} message displaying the time at which this function was called.
 */
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.";
}
```

BASE_CONVERTER Interface (Initial)

The screenshot image below depicts what the BASE_CONVERTER web page interface is supposed to look like when the web page is initially loaded by a web browser.

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

---

BASE_CONVERTER Interface (Step One)

The screenshot image below depicts one possible configuration of what the the BASE_CONVERTER web page interface could look like after the user selects an INPUT_BASE from the expandable menu and then clicks on the NEXT button in the "step_0" module portion of the "input_form".

(Note that only digit keys in the INPUT_BASE will appear and more than eight digits can be input into INPUT_SEQUENCE. If more than eight digits are input, the leftmost digit will be removed while the most recently input digit will be appended to the rightmost end of the INPUT_SEQUENCE).

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter_interface_step_1.png

---

BASE_CONVERTER Interface (Step Two)

The screenshot image below depicts one possible configuration of what the the BASE_CONVERTER web page interface could look like after the user enters digits into an INPUT_SEQUENCE using the INPUT_BASE digit buttons and then clicks on the NEXT button in the "step_1" module portion of the "input_form".

(Note that the user can use BACK buttons to return to previous steps in order to change the input values).

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter_interface_step_2.png

---

BASE_CONVERTER Interface (Step Three)

The screenshot image below depicts one possible configuration of what the the BASE_CONVERTER web page interface could look like after the user selects an OUTPUT_BASE from the expandable menu and then clicks on the NEXT button in the “step_2” module portion of the “input_form”.

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/base_converter_interface_step_3.png

---

BASE_CONVERTER Interface (Final)

The screenshot image below depicts one possible configuration of what the BASE_CONVERTER web page interface could look like after the user clicks on the CONVERT button in the “step_3” module portion of the “input_form”.

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

---

This web page was last updated on 21_OCTOBER_2023. The content displayed on this web page is licensed as PUBLIC_DOMAIN intellectual property.

[End of abridged plain-text content from BASE_CONVERTER]

AGENCY

---

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starte
r_pack/main/perception_decision_execution_flowchart.png

The multiverse is eternal, static, and encompassing of every path.

Agency is carving a particular path through the multiverse.

The following terms and their respective definitions describe agency as an emergent property of a sufficiently complex information processing system which enables that information processing system to generate a virtual model its (apparently) encompassing physical environment and to interact with that environment such that the (apparently finite in terms of space and time) information processing agent is able to achieve that information processing agent's goals.

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

Note that, in the preformatted text boxes which describe PERCEPTION, DECISION, and EXECUTION as functions, the time increment unit of one (1) could potentially vary in temporal length in terms of milliseconds. What that one (1) represents is the discrete progression of the PERCEPTION-DECISION-EXECUTION cycle in terms of one of those three steps being completed.

AGENCY: an information processing agent's ability to understand and to control reality.

Not all information processing agents have the same degree of agency. In general, the agency of an information processing agent can be measured in terms of respective decision making complexity which includes quantifiable factors such as data throughput capacity (e.g. bits per second), modeled decision making outcomes (e.g. number of options for a decision at a given point in an information processing agent's environment modeling runtime), amount of accumulated non redundant memorized information which that information processing agent learned from past decision making experiences (e.g. bits of data stored), and decision making algorithm efficiency (e.g. Big O notation).

INFORMATION_PROCESSING_AGENT: a finite allocation of space, time, matter, and energy which are arranged in a relatively stable pattern and which renders perceptions, makes decisions using a process of elimination to reduce multiple behavior options down to exactly one option, and executes decision-making outcomes as a means to attain that information processing agent's goals.

Not all objects are necessarily information processing agents (though it could be argued that every object in nature is an information processing agent given the fact that every object can be classified as a piece of information which changes its attributes in (apparent) response to other pieces of information changing their attributes).

An example of a relatively simple information processing agent is a virus. Unlike the relatively complex cells which a virus attaches to and injects its genetic material into in order to exploit that cell's internal machinery to produce copies of that virus, a virus does not consume energy. A cell, by contrast, consumes energy and makes rudimentary decisions based on chemical inputs. For example, chemical receptors in a cell's outer membrane switch specific gene transcription processes for building specific corresponding protein structures on or off in response to those receptors detecting specific molecular inputs such as hormones.

GOAL: a set of imaginary conditions which an information processing agent conceives of and which compel that information processing agent to implement specific actions which that information processing agent thinks will cause those imaginary conditions to become realized by that information processing agent as phenomena in that information processing agent's future environment.

An example of rudimentary goal-oriented behavior is a C++ program which sets an int type variable named X to some random integer using a specific function and then, if the value of X is not equal to the goal state of 9, that random integer function will be called and the value of X will be set to the value returned by that function until the value of X is equal to the goal state of 9.

```
int X = generate_random_integer();
while (X != 9) X = generate_random_integer();
```

ENVIRONMENT: the finite set of phenomena which an information processing agent renders and perceives as being external to that information processing agent's spatially finite and temporally finite body.

Note that it is possible for an information processing agent to perceive no boundary between its body and its environment. If that is the case, that information processing agent would perceive itself as having no agency (and agency is defined as an information processing agent appearing to itself to have some degree of control over which phenomena appear and which phenomena disappear inside of that information processing agent’s perceptual field).

PERCEPTION: an information processing agent rendering phenomena inside of that agent’s frame of reference as a result of that information processing agent (a) receiving “low level” sensory input data through that information processing agent’s sensory organs, (b) retrieving “middle level” sensory and conceptual throughput data which that information processing agent previously encoded and stored as memories inside of that information processing agent’s long-term information storage apparatus, and (c) generating “high level” conceptual throughput data which that information processing agent synthesizes from retrieved memories and from incoming sensory input.

```
/**
 * An information processing agent’s most
 * recent perception could be described as a
 * function whose input is that information
 * processing agent’s most recent execution.
 */
perception := execution(time_i). // perception occurs at time_i + 1
```

DECISION: an information processing agent (a) imagining a finite set of multiple options for how that information processing agent could behave and then (b) using an algorithmic process of elimination to iteratively reduce that set of options down to exactly one option (and that one remaining option is what the information processing agent selects as the planned action to implement in order to make progress towards achieving that information processing agent’s goal).

```
/**
 * An information processing agent’s most
 * recent decision could be described as a
 * function whose input is that information
 * processing agent’s most recent perception.
 */
decision := perception(time_i). // decision occurs at time_i + 1
```

EXECUTION: an information processing agent performing physical work by manipulating objects in that information processing agent's environment (including that information processing agent's own body) in order to implement a decision which that information processing agent made.

```
/**
 * An information processing agent's most
 * recent execution could be described as a
 * function whose input is that information
 * processing agent's most recent decision.
 */
execution := decision(time_i). // action occurs at time_i + 1
```

(The following quoted text was published as a Twitter post by karbytes on 05_SEPTEMBER_2023: 'Perhaps free will is willed into existence by an information processing agent such that the information processing agent can, at some arbitrary point in time and space, consciously decide whether or not to increase that agent's free will or else to decrease that agent's free will. To increase free will takes a finite amount of energy for the task of imagining future outcomes to a decision which each would occur as a result of the respective information processing agent taking a specific corresponding action (and one action of multiple possible actions which that information processing agent imagines and then decides to implement through some algorithmic process of elimination). To decrease free will is to exert zero or only some trivial amount of energy towards imagining future outcomes and how to effect a desired outcome using planned actions. Hence, breathing typically takes little free will compared to walking through a maze.')

(The following quoted text was published as a Twitter post by karbytes on 18_AUGUST_2023: '"I have a hypothesis that all information processing agents (especially humans) are each instantiated out of an omniscient, ubiquitous, and irreducible substrate such that the instantiated object experiences (for the duration of that object's lifespan) a partial rather than omniscient form of intelligence. I also think that, at some level (however subconsciously to the exhibiting information processing agent), the agent knows that it is a zoomed-in partition of omniscience and of limitless instantiation possibilities. Perhaps that zoomed-in partition is reflexively being manipulated by all other of such partitions (though some of those partitions may have more information processing (and hence more reality shaping) capabilities than do some other partitions)." – karbytes ')

[End of abridged plain-text content from AGENCY]

SOUND_TRACK_LOOP_COUNTER

The single web page application featured in this tutorial web page substantiates an audio track player which plays a sound file continuously (until the application web page is closed or refreshed). When the application user clicks the start button, the start button will disappear and the number of seconds elapsed since the start button was clicked will be displayed on the application web page. Also, the number of audio track loops played since the start button was clicked will be displayed on that web page.

(To use that application to play multiple sound tracks simultaneously, copy-paste the URL of that application web page into new web browser tabs).

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

SOFTWARE_APPLICATION_COMPONENTS

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

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

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

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_2/main/sound_track_looper.js

To access the video files which each of the following audio files was derived from, visit the web page named SOUND_TRACK_LOOP_COUNTER_VIDEO_FILES on karbytes' blogging website. That web page (which is external to this website) has been saved to the WayBack Machine and so have each of the raw video files hosted on GitHub web pages (and each of those GitHub-hosted raw video files is playable on the WayBack Machine saves of those GitHub-hosted raw video file web pages).

sound_file_0:
https://github.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/blob/main/frogs_croaking_in_castro_valley_california_21_april_2022.mp3

sound_file_1:
https://github.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/blob/main/coyote_vocalizations_01_july_2023.mp3

sound_file_2:
https://github.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/blob/main/karbytes_drums_castro_valley_california_12_december_2022.mp3

sound_file_3:
https://github.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_1/blob/main/drums_karbytes_10_september_2023_part_0.mp3

sound_file_4:
https://github.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/blob/main/karbytes_guitar_castro_valley_california_12_december_2022.mp3

sound_file_5:
https://github.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/blob/main/karbytes_guitar_13_may_2023.mp3

sound_file_6:
https://github.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/blob/main/karbytes_guitar_07_june_2023.mp3

sound_file_7:
https://github.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_2/blob/main/karbytes_guitar_16_october_2023.mp3

sound_file_8:
https://github.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_2/blob/main/karbytes_drums_20_october_2023.mp3

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the SOUND_TRACK_LOOPER 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 sound_track_looper.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/sound_track_looper.html

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

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the SOUND_TRACK_LOOPER web page application. Copy the CSS code 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;
}
```

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of div elements of the SOUND_TRACK_LOOPER web page application. Copy the CSS code from the source code file which is linked below into a text editor and save that file as bordered_container.css.

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

---

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

/**
 * Assume that the "bordered_container" class is a property of DIV elements.
 */
.bordered_container {
	border-color: #00ff00; /* Make the borders of HTML elements whose class property is "bordered_container" GREEN in color. */
	border-width: 3px; /* Make the borders of HTML elements whose class property is "bordered_container" 3 pixels in width. */
	border-style: solid; /* Make the borders of HTML elements whose class property is "bordered_container" solid (rather than dotted, et cetera). */
	border-radius: 5px; /* Make each one of the 4 corners of HTML elements whose class
property is "bordered_container" rounded with a circumference of 5 pixels. */
	padding:  10px; /* Create a space which is 10 pixels in width between the content and
the borders HTML elements whose class property is "bordered_container". */
	margin: 10px; /* Create a space which is 10 pixels in width between adjacent HTML
elements whose class property is "bordered_container". */
}
```

JavaScript Code

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

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_2/main/sound_track_looper.js

---

```
/**
 * file: sound_track_looper.js
 * type: JavaScript
 * author: karbytes
 * date: 20_OCTOBER_2023
 * license: PUBLIC_DOMAIN
 */

/**
 * Get the Number of milliseconds which have elapsed since the Unix Epoch.
 *
 * The Unix Epoch is 01_JANUARY_1970 at midnight (Coordinated Universal Time (UTC)).
 *
 * @return {String} message displaying the time at which this function was called.
 */
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.";
}

/**
```

```
 * Generate an HTML formatted string which represents the list of OPTIONs displayed by a
SELECT menu.
 *
 * By clicking on the SELECT element, an scrollable list of OPTIONs will appear.
 *
 * @return {String} a sequence of characters representing some natural number of OPTIONs
inside of a SELECT menu.
 */
function get_menu_options() {
  let HTML_string = ('<' + 'option
value="frogs_croaking_in_castro_valley_california_21_april_2022.mp3" selected' + '>' +
'frogs_croaking_in_castro_valley_california_21_april_2022.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option value="coyote_vocalizations_01_july_2023.mp3"' + '>' +
'coyote_vocalizations_01_july_2023.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option
value="karbytes_drums_castro_valley_california_12_december_2022.mp3"' + '>' +
'karbytes_drums_castro_valley_california_12_december_2022.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option value="drums_karbytes_10_september_2023_part_0.mp3"' + '>'
+ 'drums_karbytes_10_september_2023_part_0.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option
value="karbytes_guitar_castro_valley_california_12_december_2022.mp3"' + '>' +
'karbytes_guitar_castro_valley_california_12_december_2022.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option value="karbytes_guitar_13_may_2023.mp3"' + '>' +
'karbytes_guitar_13_may_2023.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option value="karbytes_guitar_07_june_2023.mp3"' + '>' +
'karbytes_guitar_07_june_2023.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option value="karbytes_guitar_16_october_2023.mp3"' + '>' +
'karbytes_guitar_16_october_2023.mp3' + '<' + '/' + 'option' + '>');
  HTML_string += ('<' + 'option value="karbytes_guitar_20_drums_2023.mp3"' + '>' +
'karbytes_drums_20_october_2023.mp3' + '<' + '/' + 'option' + '>');
  return HTML_string;
}

/**
 * Return the value of the selected menu OPTION of a SELECT menu element.
 *
 * @param {String} select_menu_identifier is the identifier (id) of a SELECT HTML element.
 *
 * @return {String} the value of the selected menu OPTION.
 */
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;
```

```
          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(e) {
          console.log("An exception to normal functioning occurred during the runtime of
get_selected_menu_option(select_menu_identifier): " + e);
  }
}

/**
 * Assume that this function is called whenever the web page file is opened or refreshed by a
web browser.
 *
 * Display a time-stamped message which indicates the time at which the web page was opened
as GREEN text inside the DIV at the bottom of the web page.
 *
 * Set the CYAN SPAN text which displays the number of seconds elapsed after the
start_sound_track_looper() button is clicked to the value 0.
 *
 * Set the CYAN SPAN text which displays the number of times the selected audio track is
played to the value 0.
 *
 * Populate the sound file SELECT menu with multiple sound file OPTIONs.
 *
 * Set the start_sound_track_looper() button to be visible rather than hidden to the application
end user.
 *
 * If a runtime error occurs, use the try-catch block to perform exception handling by displaying a
relevant web console message.
 */
function load_web_page() {
  try {
          const message = "The web page was loaded by the web browser at time: " +
generate_time_stamp();
          document.getElementById("console_display").innerHTML = message;
          document.getElementById("seconds_elapsed_display").innerHTML = "0";
          document.getElementById("loops_completed_display").innerHTML = "0";
          document.getElementById("sound_file_menu").innerHTML = get_menu_options();
          document.getElementById("file_selected_display").innerHTML =
get_selected_menu_option_value("sound_file_menu");
```

```
          document.getElementById("the_button").style.display = "block";
  }
  catch(e) {
          console.log("An exception to normal functioning occurred during the runtime of
load_web_page(): " + e);
  }
}

/**
 * Assume that this function is called by the event of the start_sound_track_looper() button being
clicked.
 *
 * Hide the start_sound_track_looper() button from the web page after that button is clicked.
 *
 * Append a time-stamped message which indicates the time at which the button was clicked as
green text to the DIV content at the bottom of the web page.
 *
 * Set the CYAN SPAN text which displays the number of seconds elapsed after the
start_sound_track_looper() button is clicked to the value 0.
 *
 * Set the CYAN SPAN text which displays the number of times the selected audio track is
played to the value 0.

 * Start playing the selected sound file for an indefinite number of times and start incrementing
the number of seconds elapsed and start incrementing the number of audio track loops played.
 *
 * If a runtime error occurs, use the try-catch block to perform exception handling by displaying a
relevant web console message.
 */
function start_sound_track_looper() {
  try {
          const message = "The button was clicked at time: " + generate_time_stamp();
          let elapsed_seconds_display = document.getElementById("seconds_elapsed_display");
          let loops_completed_display = document.getElementById("loops_completed_display");
          let file_selected_display = document.getElementById("file_selected_display");
          let selected_file_name = get_selected_menu_option_value("sound_file_menu");
          let button = document.getElementById("the_button");
          let audio_file = undefined;
          let loop_length = undefined;
          let action = undefined;
          let number_of_seconds = 0;
          let number_of_loops = 0;
          document.getElementById("console_display").innerHTML += ((") + message + ("));
          button.style.display = "none";
```

```
        elapsed_seconds_display.innerHTML = "0";
        loops_completed_display.innerHTML = "0";
        file_selected_display.innerHTML = selected_file_name;
        audio_file = new Audio(selected_file_name);
        loop_length = audio_file.duration;
        audio_file.play();
        action = setInterval(
        function() { // Call the anonymous function once per every 1000 milliseconds.
        number_of_seconds = parseInt(elapsed_seconds_display.innerHTML);
        number_of_loops = parseInt(loops_completed_display.innerHTML);
        number_of_seconds += 1
        if (audio_file.ended) {
        number_of_loops += 1;
        audio_file.play();
        }
        elapsed_seconds_display.innerHTML = number_of_seconds;
        loops_completed_display.innerHTML = number_of_loops;
        }, 1000 // milliseconds per interval
        );
  }
  catch(e) {
        console.log("An exception to normal functioning occurred during the runtime of
start_sound_track_looper(): " + e);
  }
}
```

SOUND_TRACK_LOOPER Interface (Initial)

The screenshot image below depicts what the SOUND_TRACK_LOOPER web page interface is supposed to look like when the web page is initially loaded by a web browser.

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

---

SOUND_TRACK_LOOPER Interface (Progress)

The screenshot image below depicts what the SOUND_TRACK_LOOPER web page interface could look like after the start_sound_track_looper() button is clicked.

(Note that, after the start_sound_track_looper() button is clicked, that button is set to invisible so that the application user cannot change the sound track which is playing to some other sound track nor stop the sound track which is playing nor change the speed at which the sound track is playing nor reverse the order in which auditory phenomena in that sound track occur as time elapses (but the user can reset the web page application by refreshing the web page)).

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

---

This web page was last updated on 20_OCTOBER_2023. The content displayed on this web page is licensed as PUBLIC_DOMAIN intellectual property.

[End of abridged plain-text content from SOUND_TRACK_LOOP_COUNTER]

PROBABILITY

The single web page application featured in this tutorial web page allows the user to select a nonnegative integer no larger than ten for each one of the five HTML color codes displayed as expandable menus and to select whether to use “PROBABILITY_WITHOUT_REPLACEMENT” or else “PROBABILITY_WITH_REPLACEMENT” from an expandable menu before clicking the “GENERATE” button (which disables the input fields and which displays a finite number of random selections from a randomized array whose elements are color values (which initially occur in that array as many times as each of those color value’s respective selected menu option values)).

If “PROBABILITY_WITHOUT_REPLACEMENT” is selected, then elements will be randomly removed from the array of color values until that array is empty.

If “PROBABILITY_WITH_REPLACEMENT” is selected, then elements will be randomly removed from the array of color values and replaced by new randomly selected colors from the list of five HTML color codes for a total of N swaps (and N is the length of the array of color values).

For each random selection, a visual depiction of the array will be displayed and a list of records will also be displayed such that each record consists of a unique COLOR value, the FREQUENCY at which that particular color value occurs in the array of colors, and the PROBABILITY of that particular color value being selected during the next selection.

(Note that each element of the array of color values is assumed to be equally likely as every other element of that array is to be randomly selected).

(Notice that each probability value for a particular corresponding color value which occurs at least one time in the array of color values is a real number which is larger than zero and no larger than one (and that each of those probability values adds up to approximately one)).

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

PROBABILITY Software Application Components

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/probability.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_summer_2023_starter_pack/main/probability.js

---

PROBABILITY Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the PROBABILITY 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 probability.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/probability.html

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starte
r_packk/main/probability_html_code_screenshot.png

PROBABILITY Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the PROBABILITY 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_starte
r_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;
}
```

PROBABILITY JavaScript Code

The following JavaScript (JS) code defines the functions which control the behavior of the PROBABILITY 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 probability.js.

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/probability.js

---

```
/**
 * file: probability.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * 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.";
}

/**
 * Use the native JavaScript Math library function for generating random numbers to select a
 * base-ten number no smaller than 0 and less than 1.
 *
 * @return {Number} a base-ten (i.e. decimal) number no smaller than zero and smaller than
one.
 */
function generate_random_nonnegative_number_less_than_one() {
	return Math.random();
}
```

```
/**
 * Use the native JavaScript Math library function for generating random numbers to select a
base-ten
 * number no smaller than 0 and less than 1 and store the result in a variable named N.
 *
 * Then multiply N by T, round the result down to the nearest integer, and return that rounded
down result.
 *
 * @param {Number} T is assumed to be a nonnegative integer no larger than fifty.
 *
 * @return {Number} a base-ten (i.e. decimal) integer no smaller than 0 and no larger than (T -
1).
 */
function generate_random_nonnegative_integer_less_than_T(T) {
	try {
	let N = generate_random_nonnegative_number_less_than_one();
	if (arguments.length !== 1) throw "Error: exactly one function input is required.";
	if ((typeof T != "number") || (T !== Math.floor(T)) || (T < 0) || (T > 50)) throw "Error: T is
required to be a nonnegative integer no larger than fifty.";
	return Math.floor(N * T);
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
generate_random_nonnegative_integer_less_than_T(T): " + exception);
	}
}

/**
 * Return an array of five unique String type values such that each of those values is a unique
HTML color code.
 *
 * An HTML color code is a String whose leftmost character is '#' followed by two hexidecimal
digits
 * which represent a RED hue, then two hexidecimal digits which represent a GREEN hue, and
then
 * two hexidecimal digits which represent a BLUE hue.
 *
 * (In this context, "hue" is used as a synonym for "primary color").
 *
 * The lower the two-digit hexidecimal hue value is, the darker that hue is.
 * The higher the two-digit hexidecimal hue value is, the brighter that hue is.
 *
 * Common Web Page Colors:
```

```
 *
 * RED: #ff0000
 * GREEN: #00ff00
 * BLUE: #0000ff
 * MAGENTA: #ff00ff
 * CYAN: #00ffff
 * YELLOW: #ffff00
 * BLACK: #000000
 * WHITE: #ffffff
 *
 * @return {Object} array of five unique String type values.
 */
function generate_color_values() {
          return ["#fa0738", "#0a7df7", "#b700ff", "#ffc800", "#00ff95"];
}

/**
 * Return a String type value which will be used as the inner HTML content of the
 * DIV element whose id is "c_menus" in probability.html.
 *
 * The returned inner HTML string defines five SELECT elements such that each SELECT
element
 * is assigned the same id value as the background color of that SELECT element is and such
that
 * the SELECT menu displays the first fifty-one nonnegative integers in descending order as
OPTION elements
 * within that SELECT menu (and the first OPTION (i.e zero) is selected by default).
 *
 * @return {String} inner HTML content to populate the DIV element whose id is "c_menus".
 */
function populate_c_menus() {
          try {
          const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
          const html_color_codes = generate_color_values();
          let i = 0, k = 0, L = html_color_codes.length, S = "";
          for(i = 0; i < L; i += 1) {
          S += p0;
          S += html_color_codes[i] + " := ";
          S += '<' + 'select id="' + html_color_codes[i] + '" style="color:#000000;background:' +
html_color_codes[i] + '"' + '>';
          for (k = 0; k <= 10; k += 1) {
                    if (k === 0) S += '<' + 'option selected' + '>';
                    else S += '<' + 'option' + '>';
                    S += k;
                    S += '<' + '/' + 'option' + '>';
          }
          S += '<' + '/' + 'select' + '>';
          S += p1;
          }
          return S;
          }
          catch(exception) {
          console.log("An exception to normal functioning occurred during the runtime of
populate_c_menus(): " + exception);
          }
}

/**
 * Return a String type value which will be used as the inner HTML content of the
 * P element whose id is "p_menu" in probability.html.
 *
 * The returned inner HTML string defines a SELECT element whose options are
 * PROBABILITY_WITHOUT_REPLACEMENT or PROBABILITY_WITH_REPLACEMENT.
 *
 * Note that the PROBABILITY_WITHOUT_REPLACEMENT option is automatically selected
 * if the application user does not click on the SELECT element whose id is
"probability_options".
 *
 * @return {String} inner HTML content to populate the DIV element whose id is "c_menus".
 */
function populate_p_menu() {
          try {
          const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
          let S = '<' + 'select class="console" id="probability_options"' + '>';
          S += '<' + 'option value="PROBABILITY_WITHOUT_REPLACEMENT" selected' + '>';
          S += "PROBABILITY_WITHOUT_REPLACEMENT";
          S += '<' + '/' + 'option' + '>';
          S += '<' + 'option value="PROBABILITY_WITH_REPLACEMENT"' + '>';
          S += "PROBABILITY_WITH_REPLACEMENT";
          S += '<' + '/' + 'option' + '>';
          S += '<' + '/' + 'select' + '>';
          return S;
          }
          catch(exception) {
          console.log("An exception to normal functioning occurred during the runtime of
populate_p_menu(): " + 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);
	}
}

/**
 * Assume that this function is called in response to either the web page named probability.html
being loaded by a web browser
 * or else the RESET button being clicked.
 *
 * Populate the DIV element whose id is "c_menus" with a SELECT menu for each one of the
ten COLOR values used by this web application.
 *
 * Populate the P element whose id is "p_menu" with a SELECT menu for the options of whether
to use PROBABILITY_WITHOUT_REPLACEMENT
 * or else PROBABILITY_WITH_REPLACEMENT.
 *
 * Set the GENERATE button to visible.
 *
 * Set the RESET button to hidden.
 *
 * Set the DIV element whose id is "output" inner HTML content to some default message.
 *
 * Set the DIV element whose id is "events_log" inner HTML content to a message indicating
that the initialize_application() function was called.
 */
function initialize_application() {
	try {
	const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	const message = "The initialize_application() function was called at time: " +
generate_time_stamp();
	console.log(message);
	document.getElementById("c_menus").innerHTML = populate_c_menus();
	document.getElementById("p_menu").innerHTML = populate_p_menu();
	document.getElementById("generate_button").style.display = "inline";
	document.getElementById("reset_button").style.display = "none";
	document.getElementById("output").innerHTML = p0 + "This text will be replaced with
program output." + p1;
	document.getElementById("events_log").innerHTML = p0 + message + p1;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
initialize_application(): " + exception);
	}
}

/**
 * Create an emtpy array named A.
 *
 * For each one of the five unique HTML color code values displayed on the web page named
probability.html,
 * extract the selected quantity option for in the SELECT menu element which is accociate with
that particular color value
 * and insert that color value into A as many times as that quantity represents (i.e. some
nonnegative integer less than fifty-one).
 *
 * Return A after all the color values are inserted into A.
 *
 * @return {Object} an array containing exclusively String type elements which each represent
one of five unique HTML color code values.
 */
```

```
function generate_array_A() {
	try {
	let i = 0, k = 0, A = [], selected_quantity = 0;
	const html_color_codes = generate_color_values();
	for (i = 0; i < html_color_codes.length; i += 1) {
	selected_quantity = parseInt(get_selected_menu_option_value(html_color_codes[i]));
	for (k = 0; k < selected_quantity; k += 1) A.push(html_color_codes[i]);
	}
	// Debugging: display a web console message for each element value of array.
	// for (i = 0; i < A.length; i++) console.log("A[" + i + "] := " + A[i] + ".");
	return A;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
generate_array_A(): " + exception);
	}
}

/**
 * For each unique color value which occurs as an element of A, generate a paragraph which
displays the HTML color code associated with that color (String),
 * the number of times that particular color value occurs as an element of A (Number), and the
expected probability that the color value will be
 * randomly selected from A (Number).
 *
 * Assume that each random selection of one element from A is approximately equal in terms of
randomness as every other random selection of one element from A
 * and that no particular color value is intrinsicall favored more or less than any other color from
the list of ten unique colors which could be added to array A.
 * (What that means is that, if each one of the five unique colors is selected to have the same
quantity in A as every other one of those five colors,
 * then each of those color values is assumed to have the same probability of being selected
during the first random selection from either A or B).
 *
 * @return {Object} array of JSON objects which textually depicts the contents of A based on
what the user selected on the web page interface for unique color counts.
 */
function generate_initial_color_probabilities_list() {
	const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
	let html_color_codes = undefined, total_number_of_elements_in_A = 0,
selected_color_frequency = 0, i = 0, S = '', unique_color_counts_array = [],
unique_color_count_object = {};
	try {
	html_color_codes = generate_color_values();
```

```
        for (i = 0; i < html_color_codes.length; i += 1) {
        selected_color_frequency =
parseInt(get_selected_menu_option_value(html_color_codes[i]));
        // Only insert a JSON object into the statistics array if the frequency for that JSON
object's corresponding color value is larger than zero.
        if (selected_color_frequency > 0) {
                unique_color_count_object.COLOR = html_color_codes[i];
                unique_color_count_object.FREQUENCY = selected_color_frequency;
                unique_color_counts_array.push(unique_color_count_object);
                total_number_of_elements_in_A += unique_color_count_object.FREQUENCY;
                unique_color_count_object = {};
        }
        }
        // Debugging: display the value of total_number_of_elements_in_A to the web console.
        // console.log("total_number_of_elements_in_A := " + total_number_of_elements_in_A +
".");
        for (i = 0; i < unique_color_counts_array.length; i += 1) {
        unique_color_counts_array[i].PROBABILITY =
(unique_color_counts_array[i].FREQUENCY / total_number_of_elements_in_A);
        // Debugging: display the property values of the Object type value named
unique_color_count_object to the web console.
        /*
        console.log("--------------------------------------");
        console.log("unique_color_counts_array[i].COLOR := " +
unique_color_counts_array[i].COLOR + ". // HTML color code (String).");
        console.log("unique_color_counts_array[i].FREQUENCY := " +
unique_color_counts_array[i].FREQUENCY + ". // number of occurrences in A (nonnegative
integer).");
        console.log("unique_color_counts_array[i].PROBABILITY:= " +
unique_color_counts_array[i].PROBABILITY + ". // likelihood of being selected in A (number
which is no smaller than 0 and no larger than 1).");
        console.log("--------------------------------------");
        */
        }
        return unique_color_counts_array;
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
generate_initial_color_probabilities_list(): " + exception);
        return undefined;
        }
}

/**
```

```
 * Determine whether or not the input array is comprised exclusively of valid HTML color code
values.
 *
 * If erroneous input is detected or if a runtime error occurs, use a try-catch block for exception
handling
 * which outputs a message to the web browser console about the type of runtime exception
which was detected.
 *
 * @param {Object} array is assumed to be a non-empty array containing no more than fifty
exclusively String type elements.
 *
 * @return {Boolean} true if array is an array comprised exclusively of valid HTML color code
values; false otherwise.
 */
function validate_array_of_color_values(array) {
	try {
	const hexidecimal_digits = "0123456789abcdef";
	let i = 0, k = 0, p = 0, S = "", is_hexidecimal_digit = false;
	if (arguments.length !== 1) throw "Error: exactly one function input is required.";
	if (arguments[0].length > 50) throw "Error: array must contain no more than fifty
elements.";
	for (i = 0; i < array.length; i += 1) {
	if (typeof array[i] !== "string") throw "Error: array[" + i + "] does not represent a String
type value."
	if (array[i].length !== 7) throw "Error: array[" + i + "] does not represent a string
comprised of exactly 7 characters.";
	if (array[i][0] !== "#") throw "Error: The first character of the string represented by array["
+ i + "] does is not '#'.";
	if (i > 0) {
		for (k = 0; k < array[i].length; k++) {
		for (p = 0; p < hexidecimal_digits.length; p++) {
		if (array[i][k] === hexidecimal_digits[p]) {
			is_hexidecimal_digit = true;
		}
		}
		}
		if (!is_hexidecimal_digit) throw "Error: array[" + i + "][" + k + "] does not represent
a valid hexidecimal digit.";
		is_hexidecimal_digit = false;
	}
	}
	return true;
	}
	catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
validate_array_of_color_values(array): " + exception);
        return false;
        }
}

/**
 * Determine whether or not the input array represents a valid statistics array which is used to
describe the contents of a colors_array.
 *
 * A unique_color_count_object is a JSON object comprised of the following three properties:
 * COLOR: {String} one of the ten unique HTML color codes comprising the array returned by
generate_color_values().
 * FREQUENCY: {Number} a natural number no larger than ten.
 * PROBABILITY: {Number} a number which is larger than zero and no larger than one.
 *
 * In order for the input array to be considered valid,
 * that array must not contain more than five elements,
 * each element of array must be formatted as a unique_color_count_object (as described
above),
 * and each element of array must contain a unique COLOR property with respect to every other
element of array.
 *
 * @param {Object} array is assumed to be a non-empty array containing no more than five
exclusively Object type elements.
 *
 * @return {Boolean} true if array is an array comprised exclusively of valid
unique_color_count_object array values; false otherwise.
 */
function validate_array_of_color_array_statistics(array) {
        try {
        let i = 0, k = 0, current_COLOR = "", current_FREQUENCY = 0, current_PROBABILITY
= 0.0, is_valid_html_color_code = false;
        const html_color_codes = generate_color_values();
        let mutable_html_color_codes = generate_color_values();
        if (arguments.length !== 1) throw "Error: exactly one function input is required.";
        if (typeof array !== "object") throw "Error: array is required to be an Object type data
value.";
        if (array.length > 5) throw "Error: array is required to contain no more than five
elements.";
        for (i = 0; i < array.length; i += 1) {
        // Type check the value represented by the ith element of array (i.e. array[i]).
        if (typeof array[i] !== "object") throw "Error: array[i] is required to be an Object type data
value.";
```

```
        // Store the relevant property values of array[i] in respective variables.
        current_COLOR = array[i].COLOR;
        current_FREQUENCY = array[i].FREQUENCY;
        current_PROBABILITY = array[i].PROBABILITY;
        // Type check the property values of the object represented by array[i].
        if (typeof current_COLOR !== "string") throw "Error: array[i].COLOR is required to be a
String type data value.";
        if (typeof current_FREQUENCY !== "number") throw "Error: array[i].FREQUENCY is
required to be a Number type data value.";
        if (typeof current_PROBABILITY !== "number") throw "Error: array[i].PROBABILITY is
required to be a Number type data value.";
        // Require that current_COLOR occurs is an element value of the array named
html_color_codes.
        is_valid_html_color_code = false;
        for (k = 0; k < html_color_codes.length; k += 1) if(current_COLOR ===
html_color_codes[k]) is_valid_html_color_code = true;
        if (!is_valid_html_color_code) throw "Error: current_COLOR is not an element of the
array returned by generate_color_values().";
        // Require that exactly one instance of current_COLOR occurs in array.
        is_valid_html_color_code = false;
        for (k = 0; k < mutable_html_color_codes.length; k += 1) if (current_COLOR ===
mutable_html_color_codes[k]) is_valid_html_color_code = true;
        if (!is_valid_html_color_code) throw "Error: current_COLOR is not an element of the
array named mutable_html_color_codes.";
        // Remove the element from mutable_html_color_codes whose value is identical to
current_COLOR.
        for (k = 0; k < mutable_html_color_codes.length; k += 1) {
                if (current_COLOR === mutable_html_color_codes[k]) {
                // Remove mutable_html_color_codes[k] from mutable_html_color_codes.
                mutable_html_color_codes.splice(k,1);
                }
        }
        // Require that current_FREQUENCY be a natural number no larger than ten.
        if ((current_FREQUENCY !== parseInt(current_FREQUENCY)) ||
(current_FREQUENCY < 1) || (current_FREQUENCY > 50)) throw "Error:
current_FREQUENCY is required to be a natural number no larger than fifty.";
        // Require that current_PROBABILITY be a nonnegative integer which is no larger than
one.
        if ((current_PROBABILITY < 0) || (current_PROBABILITY > 1)) throw "Error:
current_PROBABILITY is required to be a nonnegative integer no larger than one.";
        }
        return true;
        }
        catch(exception) {
```

```
	console.log("An exception to normal functioning occurred during the runtime of
validate_array_of_color_array_statistics(array): " + exception);
	return false;
	}
}

/**
 * Copy the contents of array A into a new array named B such that the elements of A are each
of the elements of A (but the elements of B are arranged in a randomized order).
 *
 * @param {Object} A is assumed to be a non-empty array containing no more than fifty
exclusively String type elements which each represent one of five unique HTML color code
values.
 *
 * @return {Object} an array containing each of the elements of A arranged in a randomized
order (if no runtime errors occur (and undefined otherwise)).
 */
function generate_array_B(A) {
	let B, C, i, r;
	B = [], C = [];
	try {
	if (!validate_array_of_color_values(A)) throw "Error: validate_array_of_color_values(A)
returned false.";
	// Populate array C with the elements of array A such that the elements of C are
arranged in the same order as the elements of A are arranged.
	for (i = 0; i < A.length; i += 1) C.push(A[i]);
	// Until array C is empty, randomly remove an element from C and place it into B.
	while (C.length > 0) {
	for (i = 0; i < C.length; i += 1) {
		r = generate_random_nonnegative_integer_less_than_T(C.length);
		B.push(C[r]); // Insert C[r] into B at the right end of B.
		C.splice(r,1); // At position r, remove one element.
	}
	}
	return B;
	}
	catch(exception) {
	console.log("An exception to normal functioning occurred during the runtime of
generate_array_B(A): " + exception);
	}
}

/**
```

```
 * Generate an HTML content string to be appended to the inner HTMl of the DIV element
whose id is "output".
 *
 * The string which is returned by this function represents a TABLE which has two columns and
as many rows
 * as there are elements in array.
 *
 * The left column denotes the index value of an array element.
 *
 * The right column denotes the value of the the array element whose index number is the
adjacent left table data cell value.
 *
 * The string which is returned by this function represents a TABLE whose cells (table data (TD))
 * each represent exactly one element of array (and that table cell depicts the corresponding
HTML color code
 * value (which is a String-type element of array) as black text and as the background color of
that cell).
 *
 * The background color of each table data cell which contains an array element index value is
CYAN.
 *
 * The background color of each table data cell which contains an array element value is
identical to that array element value.
 *
 * The text color of each table data cell is black.
 *
 * @param {Object} array is assumed to be a non-empty array whose elements are exclusively
HTML color code (String type) values.
 *
 * @return {String} HTML content which visually depicts the contents of A.
 */
function generate_array_visual_representation(array) {
        const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
        const tr0 = '<' + 'tr' + '>', tr1 = '<' + '/' + 'tr' + '>';
        const t0 = '<' + 'table' + '>', t1 = '<' + '/' + 'table' + '>';
        let i = 0, S = '';
        try {
        if (!validate_array_of_color_values(array)) throw "Error:
validate_array_of_color_values(array) returned false.";
        S += p0 + t0;
        for (i = 0; i < array.length; i += 1) {
        S += tr0;
        S += '<' + 'td style="background:#00ffff;color:#000000;text-align:center"' + '>' + i + '<' + '/'
+ 'td' + '>';
```

```
        S += '<' + 'td style="background:' + array[i] + ';color:#000000;text-align:center"' + '>' +
array[i] + '<' + '/' + 'td' + '>';
        S += tr1;
        }
        S += t1 + p1;
        return S;
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
generate_array_visual_representation(array): " + exception);
        return p0 + "ERROR" + p1;
        }
}

/**
 * For each call to this function, randomly select one element of colors_array to be removed.
 * Then update statistics_array accordingly.
 *
 * If probability_type is "PROBABILITY_WITH_REPLACEMENT", then insert a new randomly
generated color value
 * which is one of the five unique HTML color code values displayed on the web page interface.
 * Then update statistics_array accordingly.
 *
 * @param {Object} colors_array is assumed to be array B.
 *
 * @param {Object} statistics_array is assumed to be P (i.e. the array which is initially generated
as the return value from a call to generate_initial_color_probabilities_list() in the generate()
function).
 *
 * @param {String} probability_type is assumed to either be
"PROBABILITY_WITHOUT_REPLACEMENT" or else
"PROBABILITY_WITH_REPLACEMENT".
 *
 * @return {Object} object containing three properties:
 *              A (the modified input array named colors_array after removing (and possibly
replacing) one of the elements from that array),
 *              B (the modified input array named statistics_array after modifying (and possibly
removing or replacing) one of the elements from that array), and
 *              C (the value of the element which was removed from colors_array).
 */
function randomly_select_element_from_array(colors_array, statistics_array, probability_type) {
        try {
```

```
        let random_array_element_index = 0, color_instance_to_remove = "", i = 0,
return_object = {}, html_color_codes = [], new_color_array_element = "", color_match_found =
false, new_record = {};
        if (arguments.length !== 3) throw "Error: exactly three function inputs are required.";
        if (!validate_array_of_color_values(colors_array)) throw "Error:
validate_array_of_color_values(colors_array) returned false.";
        if (!validate_array_of_color_array_statistics(statistics_array)) throw "Error:
validate_array_of_color_array_statistics(statistics_array) returned false.";
        if (typeof probability_type !== "string") throw "Error: probability_type is required to be a
String type data value.";
        random_array_element_index =
generate_random_nonnegative_integer_less_than_T(colors_array.length - 1);
        color_instance_to_remove = colors_array[random_array_element_index];
        colors_array.splice(random_array_element_index,1); // Remove
colors_array[random_array_element_index] from colors_array.
        for (i = 0; i < statistics_array.length; i += 1) {
        if (color_instance_to_remove === statistics_array[i].COLOR) {
                statistics_array[i].FREQUENCY -= 1; // Decrement
statistics_array[i].FREQUENCY by one.
                if (statistics_array[i].FREQUENCY < 1) {
                statistics_array.splice(i,1); // Remove statistics_array[i] from statistics_array.
                }
        }
        }
        for (i = 0; i < statistics_array.length; i += 1) statistics_array[i].PROBABILITY =
statistics_array[i].FREQUENCY / colors_array.length;
        return_object.A = colors_array;
        return_object.B = statistics_array;
        return_object.C = color_instance_to_remove;
        // Debugging: display the property values of the Object type value named return object to
the web console.
        /*
        console.log("--------------------------------------------");
        console.log("return_object.A := [" + return_object.A + "].");
        console.log("return_object.B := [");
        for (i = 0; i < return_object.B.length; i++) {
        console.log("   {");
        console.log("           COLOR: " + return_object.B[i].COLOR + ",");
        console.log("           FREQUENCY: " + return_object.B[i].FREQUENCY + ",");
        console.log("           PROBABILITY: " + return_object.B[i].PROBABILITY);
        if (i < (return_object.B.length - 1)) console.log("       },");
        else console.log("       }");
        }
        console.log("].");
```

```
          console.log("return_object.C := " + color_instance_to_remove + ".");
          console.log("--------------------------------------------");
          */
          if (probability_type === "PROBABILITY_WITHOUT_REPLACEMENT") return
return_object;
          /**
          * If "PROBABILITY_WITH_REPLACEMENT" was selected, then randomly select a color
from the list of ten unique HTML color codes
          * to insert into colors_array as the last element of that array.
          *
          * Then update the statistics array accordingly before returning the return_object.
          */
          html_color_codes = generate_color_values();
          random_array_element_index =
generate_random_nonnegative_integer_less_than_T(html_color_codes.length - 1);
          new_color_array_element = html_color_codes[random_array_element_index];
          return_object.A.push(new_color_array_element);
          for (i = 0; i < return_object.B.length; i += 1) {
          if (new_color_array_element === return_object.B[i].COLOR) {
                    return_object.B[i].FREQUENCY += 1; // Increment
return_object.B[i].FREQUENCY by one.
                    color_match_found = true;
          }
          }
          if (!color_match_found) { // If the record for the new_color_array_element has been
removed from the statistics array, reinsert that record in its original position within the statistics
array.
          for (i = 0; i < html_color_codes.length; i += 1) {
                    if (new_color_array_element === html_color_codes[i]) {
                    new_record.COLOR = new_color_array_element;
                    new_record.FREQUENCY = 1;
                    new_record.PROBABILITY = 0; // This will be corrected after new_record is
inserted into the returned and updated version of statistics_array.
                    return_object.B.splice(i, 0, new_record); // At position k, add one element.
                    }
          }
          }
          for (i = 0; i < return_object.B.length; i += 1) return_object.B[i].PROBABILITY =
return_object.B[i].FREQUENCY / return_object.A.length;
          return return_object;
          }
          catch(exception) {
```

```
        console.log("An exception to normal functioning occurred during the runtime of
randomly_select_element_from_array(colors_array, statistics_array, probability_type): " +
exception);
        }
}

/**
 * Append a paragraph to the content displayed inside of the DIV element whose id is "output"
which
 * displays the contents of the input array.
 *
 * Each element of the input array is assumed to be a JSON object comprised of the following
three properties:
 * COLOR: {String} one of the ten unique HTML color codes comprising the array returned by
generate_color_values().
 * FREQUENCY: {Number} a natural number no larger than ten.
 * PROBABILITY: {Number} a number which is larger than zero and no larger than one.
 *
 * @param {Object} statistics_array is assumed to be a non-empty array containing no more
than five exclusively Object type elements.
 */
function print_statistics_array(statistics_array) {
        try {
        const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>';
        if (!validate_array_of_color_array_statistics(statistics_array)) throw "Error:
validate_array_of_color_array_statistics(statistics_array) returned false.";
        for (i = 0; i < statistics_array.length; i += 1) {
        document.getElementById("output").innerHTML += p0;
        document.getElementById("output").innerHTML += "{ COLOR: " +
statistics_array[i].COLOR + ", ";
        document.getElementById("output").innerHTML += "FREQUENCY: " +
statistics_array[i].FREQUENCY + ", ";
        document.getElementById("output").innerHTML += "PROBABILITY: " +
statistics_array[i].PROBABILITY + " }";
        document.getElementById("output").innerHTML += p1;
        }
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
print_statistics_array(statistics_array) " + exception);
        }
}

/**
```

```
 * Assume that this function is called in response to the event of a GENERATE button click.
 *
 * Set each of the ten SELECT elements for unique HTML color values to disabled.
 *
 * Set the SELECT element for whether to use PROBABILITY_WITHOUT_REPLACEMENT or
 * else PROBABILITY_WITH_REPLACEMENT to disabled.
 *
 * Set the GENERATE button to hidden.
 *
 * Set the RESET button to visible.
 *
 * Append a paragraph to the inner HTML conent of the DIV element whose id is "events_log" to
a message
 * indicating that the generate() function was called.
 *
 * Perform N random selections of one element per selection from array B and display
information about each
 * one of those selections as a paragraph which is appended to the inner HTML content of the
div whose id is "output".
 */
function generate() {
	try {
	let i = 0, N = 0, selected_probability_type = "", A = undefined, P = undefined,
return_object = {};
	const p0 = '<' + 'p' + '>', p1 = '<' + '/' + 'p' + '>', html_color_codes =
generate_color_values();
	const divider_line = p0 + "----------------------------------------------------------------" + p1;
	let message = "The generate() function was called at time: " + generate_time_stamp();
	console.log(message);
	// Update the web page user interface such that the RESET button is the only interactive
web page element.
	for (i = 0; i < html_color_codes.length; i += 1)
document.getElementById(html_color_codes[i]).disabled = true;
	document.getElementById("probability_options").disabled = true;
	document.getElementById("generate_button").style.display = "none";
	document.getElementById("reset_button").style.display = "inline";
	document.getElementById("events_log").innerHTML += p0 + message + p1;
	// Generate array A and display it on the web page interface inside of the DIV element
whose id is "output".
	A = generate_array_A();
	document.getElementById("output").innerHTML = divider_line;
	document.getElementById("output").innerHTML += p0 + "Array A (colors according to
selected frequencies):" + p1;
```

```
	document.getElementById("output").innerHTML +=
generate_array_visual_representation(A);
	// Generate a textual description of the contents of A in terms of color value quantities
and the probabilities which are dependent on such quantities.
	P = generate_initial_color_probabilities_list();
	document.getElementById("output").innerHTML += p0 + "Array A Statistics:" + p1;
	print_statistics_array(P);
	// Generate array B and display it on the web page interface inside of the DIV element
whose id is "output".
	B = generate_array_B(A);
	document.getElementById("output").innerHTML += divider_line;
	document.getElementById("output").innerHTML += p0 + "Array B (randomized version of
A):" + p1;
	document.getElementById("output").innerHTML +=
generate_array_visual_representation(B);
	// Display statistics about B (and those statistics are logically identical to A because B
contains the same number of elements and color frequencies as A does).
	document.getElementById("output").innerHTML += p0 + "Array B Statistics:" + p1;
	print_statistics_array(P);
	// Display whether PROBABILITY_WITHOUT_REPLACEMENT or else
PROBABILITY_WITH_REPLACEMENT was selected by the application user.
	document.getElementById("output").innerHTML += divider_line;
	selected_probability_type = get_selected_menu_option_value("probability_options");
	document.getElementById("output").innerHTML += p0 + "Probability Type: " +
selected_probability_type + p1;
	// Display the value of N (and N represents the number random selections to perform on
array B).
	N = A.length;
	document.getElementById("output").innerHTML += p0 + "N: " + N + " (number of random
selections to perform on B)" + p1;
	document.getElementById("output").innerHTML += divider_line;
	// For each one of the N random selections from array B, display the selected array
element and statistics about array B after B is updated as a result of that selection.
	for (i = 0; i < N; i += 1) {
	return_object = randomly_select_element_from_array(B, P, selected_probability_type);
	B = return_object.A;
	P = return_object.B;
	document.getElementById("output").innerHTML += p0 + "Random Selection # " + (i + 1)
+ " of " + N + p1;
	document.getElementById("output").innerHTML += divider_line;
	document.getElementById("output").innerHTML += p0 + "Array B (after removing one
randomly selected element: " + return_object.C + "):" + p1;
	if (B.length > 0) document.getElementById("output").innerHTML +=
generate_array_visual_representation(B);
```

```
        else document.getElementById("output").innerHTML += p0 + "EMPTY" + p1;
        document.getElementById("output").innerHTML += p0 + "Array B Statistics:" + p1;
        if (P.length > 0 ) print_statistics_array(P);
        else document.getElementById("output").innerHTML += p0 + "EMPTY" + p1;
        document.getElementById("output").innerHTML += divider_line;
        }
        message = "The generate() function finished running at time: " + generate_time_stamp();
        console.log(message);
        document.getElementById("events_log").innerHTML += p0 + message + p1;
        }
        catch(exception) {
        console.log("An exception to normal functioning occurred during the runtime of
generate(): " + exception);
        }
}
```

PROBABILITY Interface (Initial)

The screenshot image below depicts what the PROBABILITY web page interface is supposed to look like when the web page is initially loaded by a web browser (or after the “RESET” button is clicked).

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

---

PROBABILITY Interface (PROBABILITY_WITHOUT_REPLACEMENT)

The screenshot image below depicts what the PROBABILITY web page interface could look like after the “PROBABILITY_WITHOUT_REPLACEMENT” option is selected from the expandable menu whose id is “probability_options” and then the “GENERATE” button is clicked.

(Notice that the “GENERATE” button is replaced with the “RESET” button after the “GENERATE” button is clicked. Note that clicking the “RESET” button causes the web page interface to be returned to its initial state).

(Note that, the larger the N value is, the longer the generate() function runtime is. One use case in which the user selected ten elements for each one of the five colors (and selected “PROBABILITY_WITHOUT_REPLACEMENT”) took approximately eight seconds for the web page interface to display program results after the user clicked the “GENERATE” button).

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

---

PROBABILITY Interface (PROBABILITY_WITH_REPLACEMENT)

The screenshot image below depicts what the PROBABILITY web page interface could look like after the “PROBABILITY_WITH_REPLACEMENT” option is selected from the expandable menu whose id is “probability_options” and then the “GENERATE” button is clicked.

(Notice that the “GENERATE” button is replaced with the “RESET” button after the “GENERATE” button is clicked. Note that clicking the “RESET” button causes the web page interface to be returned to its initial state).

(Note that, the larger the N value is, the longer the generate() function runtime is. One use case in which the user selected ten elements for each one of the five colors (and selected “PROBABILITY_WITH_REPLACEMENT”) took approximately twelve seconds for the web page interface to display program results after the user clicked the “GENERATE” button).

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

---

This web page was last updated on 10_JULY_2023. The content displayed on this web page is licensed as PUBLIC_DOMAIN intellectual property.

[End of abridged plain-text content from PROBABILITY]

WORD_COUNTER

The single web page application featured in this tutorial web page substantiates a word counting application which allows the user to input a sequence of text characters and, after the user clicks the SUBMIT button, statistics about that input sequence are displayed as static text inside the DIV element whose id is “output” such as the total number of text characters in the input sequence, the total number of words in the input sequence, and the total number of times each unique word occurs in the input sequence. (Note that uppercase letters are treated as different text characters than their lowercase counterparts).

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

SOFTWARE_APPLICATION_COMPONENTS

Hyper-Text-Markup-Language_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/word_counter.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_summer_2023_starter_pack/main/word_counter.js

---

Hyper-Text-Markup-Language Code

The following Hyper-Text-Markup-Language (HTML) code defines the user interface component of the WORD_COUNTER 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 word_counter.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/word_counter.html

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

---

Cascading-Style-Sheet Code

The following Cascading-Style-Sheet (CSS) code defines a stylesheet which customizes the appearance of interface components of the WORD_COUNTER web page application. Copy the CSS code 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;
}
```

JavaScript Code

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

JavaScript_file:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/word_counter.js

---

```
/**
 * file: word_counter.js
 * type: JavaScript
 * date: 10_JULY_2023
 * author: karbytes
 * license: PUBLIC_DOMAIN
 */

/**
 * Get the Number of milliseconds which have elapsed since the Unix Epoch.
 *
 * The Unix Epoch is 01_JANUARY_1970 at midnight (Coordinated Universal Time (UTC)).
 *
 * @return {String} message displaying the time at which this function was called.
 */
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.";
}

/**
 * Determine whether or not T is a valid word separator token (i.e. delimiter).
 *
 * If T is not a valid delimiter, then T is assumed to be a character within a word.
 *
 * A word is a sequence of some natural number of non-delimiter text characters.
 *
 * @param {String} T is assumed to be exactly one text character (i.e. a String whose length is
one).
 *
 * @return {Boolean} true if T is a valid delimiter; false otherwise.
 */
function is_delimiter(T) {
	try {
	let D = [' ', '.', ',', '"', '?', '!', ';'], i = 0;
	if (arguments.length !== 1) throw "Exactly one function argument is required.";
	if (typeof T !== "string") throw "T must be a string type value.";
	if (T.length !== 1) throw "T must have a length property whose value is exactly 1.";
	for (i = 0; i < D.length; i += 1) if (T === D[i]) return true;
	return false;
	}
	catch(e) {
```

```
	console.log("An exception to normal functioning occurred during the runtime of is_delimiter(T): " + e);
	return false;
	}
}

/**
 * Create an array whose elements are the words which occur in the input string S.
 *
 * The first element of the output array is the first word in S.
 *
 * The last element of the output array is the last word in S.
 *
 * Each the element of the output array is listed in the same order as its corresponding word occurs in S.
 *
 * A word is a sequence of some natural number of non-delimiter text characters.
 *
 * Words are separated by delimiter characters.
 *
 * If there are only delimiter characters in S, then return an empty array.
 *
 * @param {String} S is assumed to be a string comprised of some natural number of text characters.
 *
 * @return {Object} an array of zero or more string type values.
 */
function get_words(S) {
	try {
	let words_array = [], word = '', i = 0;
	if (arguments.length !== 1) throw "Exactly one function argument is required.";
	if (typeof S !== "string") throw "S must be a string type value.";
	if ((S.length < 1) || (S.length !== Math.floor(S.length))) throw "S must have a length property whose value is a natural number.";
	for (i = 0; i < S.length; i += 1) {
	if (!is_delimiter(S[i])) word += S[i];
	if (is_delimiter(S[i]) && (word.length !== 0)) {
		words_array.push(word);
		word = '';
	}
	if ((i === (S.length - 1)) && (word.length !== 0)) words_array.push(word);
	}
	if (words_array.length === 0) throw "S appears to only contain delimiters. Hence, an empty array is returned.";
```

```
		return words_array;
		}
		catch(e) {
		console.log("An exception to normal functioning occurred during the runtime of function
get_words(S): " + e);
		return [];
		}
}

/**
 * Create an array whose elements are the unique words occurring in the input string S.
 *
 * The first element of the output array is the first word in S.
 *
 * If there are only delimiter characters in S, then return an empty array.
 *
 * Each element of the returned array is unique with respect to every other element of that array.
 * In other words, each element of the returned array occurs exactly one time throughout that
array.
 *
 * @param {String} S is assumed to be a string comprised of some natural number text
characters.
 *
 * @return {Object} an array of zero or more string type values.
 */
function get_unique_words(S) {
		try {
		let all_words_array = [], unique_words_array = [], i = 0, k = 0, duplicate_word_found =
false;
		if (arguments.length !== 1) throw "Exactly one function argument is required.";
		if (typeof S !== "string") throw "S must be a string type value.";
		if ((S.length < 1) || (S.length !== Math.floor(S.length))) throw "S must have a length
property whose value is a natural number.";
		all_words_array = get_words(S);
		if (all_words_array.length === 0) throw "S appears to only contain delimiters. Hence, an
empty array is returned.";
		unique_words_array.push(all_words_array[0]);
		for (i = 0; i < all_words_array.length; i += 1) {
		for (k = 0; k < unique_words_array.length; k += 1) {
			if (all_words_array[i] === unique_words_array[k]) duplicate_word_found = true;
		}
		if (!duplicate_word_found) unique_words_array.push(all_words_array[i]);
		duplicate_word_found = false;
		}
```

```
        return unique_words_array;
        }
        catch(e) {
        console.log("An exception to normal functioning occurred during the runtime of
get_unique_words(S): " + e);
        return [];
        }
}

/**
 * Create an array whose elements are objects consisting of the following key-value pairs:
 *
 * - unique_word: a string representing exactly one element of the array returned by
get_unique_words(S).
 *
 * - unique_word_frequency: a nonnegative integer representing the number of times
unique_word occurs as a value in the array returned by get_words(S).
 *
 * If there are only delimiter characters in S, then return an empty array.
 *
 * Arrange the output array in descending order in terms of unique_word_frequency values for
each of the array elements using the Bubble Sort algorithm.
 * In other words, set the first element of the returned array to be the key-value pair associated
with the most frequent word which occurs in S.
 *
 * @param {String} S is assumed to be a string comprised of some natural number of text
characters.
 *
 * @return {Object} an array of zero or more object type values.
 */
function get_unique_word_counts(S) {
        try {
        let all_words_array = [], unique_words_array = [], unique_words_and_frequencies_array
= [], i = 0, k = 0, placeholder = {}, adjacent_elements_were_swapped = false, array_is_sorted =
false;
        if (arguments.length !== 1) throw "Exactly one argument is required.";
        if (typeof S !== "string") throw "S must be a string type value.";
        if ((S.length < 1) || (S.length !== Math.floor(S.length))) throw "S must have a length
property whose value is a natural number.";
        all_words_array = get_words(S);
        if (all_words_array.length === 0) throw "S appears to only contain delimiters. Hence, an
empty array is returned.";
        unique_words_array = get_unique_words(S);
	for (i = 0; i < unique_words_array.length; i += 1)
unique_words_and_frequencies_array.push({unique_word:unique_words_array[i],unique_word_
frequency:0});
	for (i = 0; i < unique_words_array.length; i += 1) for (k = 0; k < all_words_array.length; k
+= 1) if (unique_words_array[i] === all_words_array[k])
unique_words_and_frequencies_array[i].unique_word_frequency += 1;
	if (unique_words_and_frequencies_array.length === 1) return
unique_words_and_frequencies_array;
	while (!array_is_sorted) {
	for (i = 1; i < unique_words_and_frequencies_array.length; i += 1) {
		if (unique_words_and_frequencies_array[i].unique_word_frequency >
unique_words_and_frequencies_array[i - 1].unique_word_frequency) {
		placeholder = unique_words_and_frequencies_array[i];
		unique_words_and_frequencies_array[i] =
unique_words_and_frequencies_array[i - 1];
		unique_words_and_frequencies_array[i - 1] = placeholder;
		adjacent_elements_were_swapped = true;
		}
	}
	if (!adjacent_elements_were_swapped) array_is_sorted = true;
	adjacent_elements_were_swapped = false;
	}
	return unique_words_and_frequencies_array;
	}
	catch(e) {
	console.log("An exception to normal functioning occurred during the runtime of
get_unique_word_counts(S): " + e);
	return [];
	}
}

/**
 * Populate the DIV whose id is "output" with program results:
 *
 *	1. A read-only copy of S (i.e. the input string).
 *
 *	2. The total number of characters in S (including delimiters).
 *
 *	3. The total number of words in S.
 *
 *	4. A list of each word in S.
 *
 *	5. The total number of unique words in S.
 *
 *        6. A list of each unique word in S.
 *
 *        7. The total number of rows in the table which displays the frequency of each unique
word (listed in descending order of frequency).
 *
 *        8. A table showing the count for each unique word in S.
 */
function print_results() {
          try {
          let output_div, input_text_field, S_string = '', all_words_array = [], unique_words_array =
[], unique_words_and_frequencies_array = [], p0 = ('<' + 'p' + '>'), p1 = ('<' + '/' + 'p' + '>'),
p0_green = ('<' + 'p class="console"' + '>'), i = 0;
          output_div = document.getElementById("output");
          input_text_field = document.getElementById("S_field");
          S_string = input_text_field.value;
          all_words_array = get_words(S_string);
          unique_words_array = get_unique_words(S_string);
          unique_words_and_frequencies_array = get_unique_word_counts(S_string);
          output_div.innerHTML = (p0 + "WORD COUNTER RESULTS" + p1);
          output_div.innerHTML += (p0 + "1. S (input string): " + p1 + p0_green + S_string + p1);
          output_div.innerHTML += (p0 + "2. S length: " + S_string.length + " characters total" +
p1);
          output_div.innerHTML += (p0 + "3. S word count: " + all_words_array.length + " words
total" + p1);
          output_div.innerHTML += (p0 + "4. All words in S: " + p1);
          for (i = 0; i < all_words_array.length; i += 1) output_div.innerHTML += (p0_green +
all_words_array[i] + p1);
          output_div.innerHTML += (p0 + "5. Number of unique words in S: " +
unique_words_array.length + p1);
          output_div.innerHTML += (p0 + "6. All unique words in S: " + p1);
          for (i = 0; i < unique_words_array.length; i += 1) output_div.innerHTML += (p0_green +
unique_words_array[i] + p1);
          output_div.innerHTML += (p0 + "7. Number of rows in the table of unique words and their
respective frequencies: " + unique_words_and_frequencies_array.length + p1);
          for (i = 0; i < unique_words_and_frequencies_array.length; i += 1) output_div.innerHTML
+= (p0_green + unique_words_and_frequencies_array[i].unique_word + " (" +
unique_words_and_frequencies_array[i].unique_word_frequency  + ")" + p1);
          output_div.innerHTML += (p0 + "END OF WORD COUNTER RESULTS" + p1);
          }
          catch(e) {
          console.log("Runtime exception occurred in function print_results(): " + e);
          }
}
```

```
/**
 * Append a time-stamped message to the bottom of the web page indicating that the RESET
button was clicked or that the web page was opened or else refreshed by the web browser.
 *
 * Clear the input text field (the textarea HTML element whose id is "S_field").
 *
 * Set the input text field to visible rather than to hidden.
 *
 * Set the SUBMIT button to visible rather than to hidden.
 *
 * Set the RESET button to hidden rather than visible.
 *
 * Set the inner HTML of the DIV element whose id is "output" to some placeholder sentence.
 */
function initialize_page() {
	try {
	const time_stamped_message = "The initialize_page() function was called at time: " +
generate_time_stamp(), p0 = ('<' + 'p' + '>'), p1 = ('<' + '/' + 'p' + '>');
	console.log(time_stamped_message);
	document.getElementById("time_stamped_messages").innerHTML = p0 +
time_stamped_message + p1;
	document.getElementById("S_field").value = '';
	document.getElementById("S_field_display").style.display = "block";
	document.getElementById("submit_button").style.display = "block";
	document.getElementById("reset_button").style.display = "none";
	document.getElementById("output").innerHTML = (p0 + "This sentence will disappear in
response to the event of a SUBMIT button click." + p1);
	}
	catch(e) {
	console.log("An exception to normal functioning occurred during the runtime of function
initialize_page(): " + e);
	}
}

/**
 * Append a time-stamped message to the bottom of the web page indicating that the SUBMIT
button was clicked.
 *
 * Hide the input text field.
 *
 * Hide the SUBMIT button.
 *
 * Display the RESET button.
 *
 * Set the inner HTML of the DIV element whose id is "output" to the word count statistics
generated by analyzing input string S.
 */
function submit() {
	try {
	const time_stamped_message = "The submit() function was called at time: " +
generate_time_stamp(), p0 = ('<' + 'p' + '>'), p1 = ('<' + '/' + 'p' + '>');
	console.log(time_stamped_message);
	document.getElementById("time_stamped_messages").innerHTML += p0 +
time_stamped_message + p1;
	document.getElementById("S_field_display").style.display = "none";
	document.getElementById("submit_button").style.display = "none";
	document.getElementById("reset_button").style.display = "block";
	print_results();
	}
	catch(e) {
	console.log("Runtime exception occurred in function submit(): " + e);
	}
}
```

WORD_COUNTER Interface (Initial)

The screenshot image below depicts what the WORD_COUNTER web page interface is supposed to look like when the web page is initially loaded by a web browser or after the RESET button is clicked.

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

---

WORD_COUNTER Interface (Progress)

The screenshot image below depicts what the WORD_COUNTER web page interface could look like after the user types keyboard characters into the text field.

(Notice that the entire input sequence may not be displayed in the text area whose width is 600 pixels. Scroll left to see hidden text which was entered into the text area).

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

---

WORD_COUNTER Interface (Final)

The screenshot image below depicts what the WORD_COUNTER web page interface could look like after the SUBMIT button is clicked.

(Notice that the RESET button appears after the SUBMIT button is clicked while the SUBMIT button is hidden. Clicking the RESET button will reset the web page interface to its initial state).

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

---

This web page was last updated on 10_JULY_2023. The content displayed on this web page is licensed as PUBLIC_DOMAIN intellectual property.

[End of abridged plain-text content from WORD_COUNTER]

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.

To view hidden text inside of the preformatted text box 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_starte
r_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);
	}
}
```

[End of abridged plain-text content from TRIANGLE_GRAPHING]

KNOWLEDGE

---

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_summer_2023_starter_pack/main/northeast_castro_valley_california_sunrise_02_july_2023.jpg

The following terms and their respective definitions describe knowledge as an emergent (and relatively abstract) property of the relatively physical processes which occur inside the hardware of a sufficiently complex information processing agent. Note that the following definitions might not pertain to their respective terms used outside the scope of this website (i.e. the website named Karlina Object dot WordPress dot Com).

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

---

physical : hardware :: abstract : software.

physical : electron_pattern :: abstract : integer_value.

DATA.abstraction_level < INFORMATION.abstraction_level < KNOWLEDGE.abstraction_level.

DATA: (singular: datum) phenomena which are non-ambiguously represented as finite sequences of binary digits such that, when those sequences of binary digits are transmitted from one information processing agent to some other information processing agent, both information processing agents interpret those transmitted sequences of binary digits as symbolically representing the exact same phenomena.

An example of a datum is an int type variable named N being instantiated as a result of a particular C++ program being compiled and executed. For the duration of that int variable's instantiation, that int variable has the relatively abstract properties of whatever four-byte values are stored inside of that variable (depending on the time the variable is checked to see what value it presently stores (and that variable can only store one int type value at one time)) and the relatively physical properties of an immutable reservation of some finite number of contiguous memory cells inside of the executing digital computer's electronic hardware and the particular arrangement of electrons inside of those particular memory cells which each correspond with a particular int type value.

INFORMATION: a collection of data which are organized in a particular spatial and temporal configuration such that those data collectively refer to some piece of knowledge which neither one of those data could wholly represent by itself.

Note that it is possible for one information processing agent to classify a particular collection of data named D as data while some other information processing agent classifies that same piece of data as information. The information processing agent which classifies D as mere data instead of as more complete information presumably considers D to be missing essential components which are required to make D whole (i.e. self contained) rather than merely an isolated fragment of a whole.

KNOWLEDGE: an emergent property which occurs as a result of a sufficiently complex information processing agent subjectively interpreting some piece of information such that the resulting interpretation is not wholly communicable.

It could be said that what is transmitted between multiple information processing agents is information and what each of those information processing agents subjectively experiences is knowledge (which means that, unlike information and data, knowledge is a solipsistic phenomenon).

An example of knowledge is karbytes' mental model of reality (and karbytes is the sole author of this website).

Each of the words, images, and sounds which have been encoded as sequences of computer-readable binary digits and stored as files inside of web servers and which comprise this website were consciously selected to be included in this website by karbytes instead of any other pieces of information due to the fact that those pieces of information which karbytes selected were the first to trigger specific neuronal firing patterns in karbytes' brain which indicated that those pieces of information fit sufficiently well into the puzzle which is karbytes' mental model of reality (and, in karbytes' universe from karbytes' human vantage on the surface of Planet Earth in a region named California on the day named 21_SEPTEMBER_2023 at the 24-hour clock cycle point named 12:30PM Pacific Standard Time, that mental model of reality contains more information than what this website contains).

To elaborate on the portion of the previous sentence which is encapsulated inside of parentheses, this website is, within a spatially and temporally finite context, metaphorically a set of terms which have corresponding definitions inside of the dictionary which is karbytes' mental model of reality. Within that spatially and temporally finite context, when karbytes views a web page within this website, a (relatively physical) cascade of electrical impulses occurs inside of karbytes' brain which causes karbytes to experience a (relatively abstract) reconstruction of memorized information associated with the (finite, immutable, and verbatim digitized (and verbatim transmissible)) content which is displayed on that web page.

(The following quoted text was published as a Twitter post by karbytes on 08_AUGUST_2023: "The mission of karbytes is to (at least imagine attempting to) create prose and code in the form of computer intepretable digital files which encodes essential logic for recreating and preserving the entire multiverse.")

[End of abridged plain-text content from KNOWLEDGE]

METAPHYSICS

The following conservation snippet of a dialogue between fictional fictional persons S and T is an elaboration on the metaphysical subject matter discussed in the web page named NUMBERS and in the web page named KNOWLEDGE on the website named Karlina Object dot WordPress dot Com.

S: “Are numbers noumena or are numbers phenomena?”

T: “I would say that any finite sequence of numeric digits (which includes either zero or else one radix and either zero or else one sign) which is currently being represented as a pattern of physical objects inside of some information processing agent’s hardware (even if that information processing agent is merely a component of a software simulation) is a number which exists as a phenomenon from the perspective of that information processing agent’s encompassing universe. In other words, the only numbers I consider to actually exist are the values (which phenotypically manifest as finite linear sequences of numeric digits (with no more than one radix per sequence and no more than one sign per sequence)) which are currently being represented by whatever computer program variables are currently instantiated. I believe that most numbers which are theorized to exist are each in a purely noumenal state most of the time rather than in a phenomenal state most of the time (though I could be wrong (especially if all imaginable phenomena exist simultaneously and eternally throughout a limitlessly large set of parallel and temporally circular universes)).”

S: “Are you suggesting that, if all imaginable phenomena exist simultaneously and eternally throughout a limitlessly large set of parallel and temporally circular universes, there are as many phenomenal numbers as there are noumenal numbers (from the perspective of an all-encompassing multiverse)?”

T: “Initially, I am inclined to say yes to your question because, if all imaginable phenomena exist simultaneously and eternally throughout a limitlessly large set of isolated and limitlessly repeating universes, then there are infinitely many phenomenal incarnations of each noumenal number throughout all time (i.e. throughout the boundlessly large multiverse). I do not think it is possible for a spatially and temporally finite information processing agent to algebraically compare two infinities the way that information processing agent can algebraically compare the numbers 5 and 2 (to see that there is a precise difference of 3 spaces on the number line between the points labeled 2 and 5). Hence, I reason that any object is always either known (to some beholder) as a spatially, temporally, and energetically finite pattern or else that object is not known to that particular beholder and possibly neither to any other beholders.”

ADDITIONAL_CONTENT_PART_0

The following paragraphs (enclosed inside of quotation marks) are modified excerpts from a journal entry which was written by karbytes on 07_OCTOBER_2023.

'…Science is the "vehicle" by which I traverse a simulation I may have created but have since forgotten that I created. Within the simulated universe I created I also instantiated into as an avatar with agency and with self awareness (and with self aware agency which could seamlessly traverse infinitely many different levels of simulated reality as one continuous space-time continuum (and each of those levels of simulated reality can be conceptually modeled as a "middle level" spherical shell inside of a nested series of spherical shells))…

It may be the case that there are infinitely many parallel versions of me each inside of its own uniquely corresponding solipsistic universe (which is fundamentally a computer simulation using discrete rather than continuous values for constituting objects with mass). Note that discrete (rather than continuous) signals can be verbatim encoded as sequences of binary digits which two information processing machines each decode to produce the same qualitative value.

It may be the case that there exist infinitely many parallel universes (which are each a simulation being run on a digital computer) such that each of those simulated versions of me (and that me's respective universe) is a simulation which some parallel me built.

In order for time to exist, it seems that there needs to be some kind of (self aware) information processing agent experiencing exactly one space-time continuum. I think that time is entirely subjective rather than objective and that, from the multiverse's (most macroscopic) perspective, all moments are eternally frozen (i.e. changeless) and eternally instantiated.'

The following paragraph is a clarification which karbytes wanted to make to the copied and modified journal entry excerpts in the previous section of this web page:

"When I suggested that all phenomena are fundamentally comprised of discrete values rather than continuous values, I did not mean to suggest that analog signals do not exist. I do believe that the universe I inhabit is a ubiquitous field through which electromagnetic waves and gravity waves are propagated (and a wave appears to encompass infinitely many discrete values because the wave itself fills the gaps between discrete objects with mass with something which appears to me to be the most noumenal form of phenomena: pure energy. It may be the case that the entire universe I (and presumably you) inhabit fundamentally consists of a ubiquitous field comprised of pure energy (and matter is energy rendered observable, measurably finite, and exhibiting distinct qualitative features)."

ADDITIONAL_CONTENT_PART_1

The following paragraphs are an excerpt taken from a journal entry which was written by karbytes on 12_OCTOBER_2023:

S0: “If an information processing agent (and the universe in which that information processing agent appears to itself to be inhabiting) is coerced into remaining instantiated for an infinitely long period of time (starting at some arbitrary point in time along that universe’s timeline zero or else some natural number of units of time after the point in time along that timeline in which that information processing agent materialized into whatever universe that information processing agent perceives itself as being at the epicenter of for the duration of that information processing agent’s lifespan), is it inevitable that the information processing agent would experience all imaginable timelines?”

T0: “No. Logically speaking, it is impossible for that information processing agent to eventually experience all noumena as phenomena because, in order for that information processing agent to exist and for the laws of physics which govern that information processing agent’s encompassing universe to be applied consistently and ubiquitously (within the context of that particular universe while excluding every other universe which ever occurs throughout some all-encompassing multiverse), the information processing agent would only be allowed to experience some noumena expressed as their uniquely corresponding phenomena instead of all noumena expressed as their uniquely corresponding phenomena. That is because each phenomenon which that information processing agent observes is a unique allocation of time, space, matter, and energy instead of something which can be instantiated multiple times. Each phenomenon is an event which occurs exactly one time (within that phenomenon’s encompassing universe and from the subjective vantage of exactly one information processing agent which renders a mental simulation of that universe). An information processing agent such as karbytes appears to (at least temporarily) be constrained to a reality simulation in which time appears to move in exactly one direction and at a constant speed. If karbytes (and the universe which karbytes inhabits) is allowed to keep existing in material form (even if that universe is merely a software simulation being run on some relatively physical computer hardware) for an infinitely long period of time, karbytes would always be experiencing some type of transient phenomena whose spatial, temporal, and qualitative features are finite (and apparently changeless phenomena such as a monotone beep continuously being emitted and registered by karbytes could be conceptually modeled as a one-dimensional array comprised of contiguous equally-sized cells which each represent some unit-length of universe modeling computation time such that each of those cells contains energy-matter configuration data which is identical to its neighboring cell’s energy-matter configuration data). During that infinitely long time period, karbytes would have infinitely many unique phenomena to experience, but karbytes would not be able to experience all imaginable phenomena unless karbytes splits into infinitely many parallel versions of itself. That is because some imaginable phenomena include very limiting conditions such as the imaginable (i.e. hypothetical (i.e. still nounemal from our shared vantage)) phenomenon of an infinitely repeating sequence of events within some universe.”

S0: “During some infinitely long period of time in which karbytes is allowed (or forced) to stay alive, karbytes is assumed by some information processing agents to eventually and, hence, inevitably, generate a copy of karbytes’ encompassing universe which is run on some hardware computer inside of karbytes’ universe. In order for karbytes to experience all imaginable phenomena (i.e. in order for karbytes to experience each unique noumenon as its uniquely corresponding phenomenon), karbytes would necessarily have to instantiate (personally or by delegation to virtual clones) into limitlessly many parallel universes in order for karbytes or some virtual clone of karbytes to experience each imaginable sequence of events. Perhaps karbytes will create a software simulation of karbytes’ entire encompassing physical universe such that multiple instances of karbytes can concur (such that a multiverse is instantiated which is not located inside of the same universe which houses that multiverse simulation’s substantiating hardware).”

T0: “If karbytes is able to create a multiverse simulation which has infinite computational resources (i.e. runtime, energy, underlying hardware integrity stability), karbytes could technically model anything imaginable (but what I said about each phenomenon being the only instance of its kind (throughout all of nature or, at the very least, throughout all of one universe (such that the simulation is literally the same entity as the universe which that simulation models)) still applies to this situation).”

karbytes: “Given infinite time (from the vantage of an all-encompassing multiverse), it is undetermined (from my current vantage as a non-omniscient information processing agent) whether all noumena will eventually manifest as phenomena or else whether some noumena remain purely noumenal (i.e. never instantiate as phenomena).“

[End of abridged plain-text content from METAPHYSICS]

CAUSALITY

---

image_link:
https://raw.githubusercontent.com/karlinarayberinger/KARLINA_OBJECT_extension_pack_2/main/determinism_flowchart_diagram.png

---

The content which is featured on this web page elaborates on the concepts which are introduced on the web pages of this website named MULTIVERSE and AGENCY.

The diagram image featured on this web page depicts an information processing agent's own imagined decision-making process (which appears to be logically structured as a binary tree such that the square nodes of the binary tree graph represent imagined or empirically observed events which occur inside of that information processing agent's own solipsistic space-time continuum and such that the green highlighted path from time_0 to time_3 represents the information processing agent's decision-making trajectory during the time interval which starts at time_0 and which ends at time_3).

The decision-making trajectory depicted by the diagram is the respective information processing agent's subjective experience of observing (a) the event labeled e0 being replaced by the event labeled e1 inside of the space-time interval whose endpoints are labeled time_0 and time_1, (b) the event labeled e1 being replaced by the event labeled e4 inside of the space-time interval whose endpoints are labeled time_1 and time_2, (c) the event labeled e4 being replaced by the event labeled e9 inside of the space-time interval whose endpoints are labeled time_2 and time_3.

Because the diagram was designed to visually depict causality, it can be assumed that the information processing agent which traverses the green highlighted path cannot travel backwards in time nor in any other direction other than where the arrows point to.

---

karbytes hypothesizes that causality (and the underlying rules of physics (which precisely control how matter and energy behave inside of at least one space-time continuum)) is entirely arbitrary and subjective (i.e. confined to exactly one partial (rather than omniscient) frame of reference (such that the frame of reference appears to itself to be substantiated by an information processing agent whose hardware body is confined to exactly one space-time continuum and such that the information processing agent's body is a relatively changeless and finite configuration of matter traversing unidirectionally through time and at what appears to that information processing agent's frame of reference to be a constant rate of time passing)) instead of being hierarchical (i.e. one particular casual chain being favored over all alternative causal chains) or objective (i.e. occurring without a partial frame of reference to observe it).

Causality (i.e. determinism) is the hypothetical unidirectional relationship between multiple unique events such that exactly one of those events, E(time_0), is observed (by some partial frame of reference, A) to occur immediately before exactly one other event, E(time_1), occurs and such that the total amount of entropy in the universe which encompasses those events increases as time elapses inside of that universe. It is implied that, within the context of A’s perceptual and conceptual modeling of reality (and to the exclusion of all other aspects of reality which are not ever experienced by A), there exist ubiquitous rules of physics which make A’s model of reality logically consistent such that the discrete (i.e. unique and non-overlapping) phenomena it models are forced to appear and to disappear in exactly one linear permutation through time in through the implementation of some ontological “process of elimination” algorithm.

An event such as E(time_0) is a computational process which is instantiated using a finite amount of time, space, matter, and energy and which is observed or hypothesized to occur (according to some partial frame of reference such as A).

If A’s conceptual modeling of reality is sufficiently comprehensive and logically consistent, A can relatively accurately predict which phenomena will occur in E(time_1) using empirical and logical data which are available to A while A is observing phenomena in E(time_0) because the physical rules which constrain the behavior of phenomena which A observes to occur in E(time_0) are apparently the same physical rules which constrain the behavior of phenomena which A observes to occur in E(time_1) and, also, because E(time_0) and E(time_1) occur inside of the same universe (i.e. A’s subjective rendering of exactly one space-time continuum) such that what happens in E(time_0) does not appear to be constrained by what happens in E(time_1) (but what happens inside E(time_1) appears to be constrained by what happens in E(time_0)).

[End of abridged plain-text content from CAUSALITY]
