Hyper Pictogramming

(February 12, 2026 edition)

Chapter 1: Introduction

Hyper Pictogramming is an application where you can learn techniques for creating web pages through making pictograms.

Section 1.1: Pictograms

A pictogram is a graphic symbol (a pictorial sign) that conveys meaning by using the shape of what it represents. Pictograms are standardized for many purposes such as guidance, safety, facilities, and equipment.

非常口
Emergency exit

SOS
非常ボタン
Emergency call button

Figure: Examples of pictograms (quoted from the Ministry of Land, Infrastructure, Transport and Tourism website)

Pictograms are used worldwide as a universal form of symbolic expression. In recent years, research on pictograms has become increasingly active, driven in part by globalization and the rapid increase in international tourists.

Section 1.2: HTML, CSS, and JavaScript

To create web pages, the technologies HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript are used. HTML defines the structure of a web page, CSS specifies its appearance, and JavaScript controls its behavior. Let’s enjoy learning by creating pictograms with Hyper Pictogramming.

Chapter 2: Access and Screen Overview

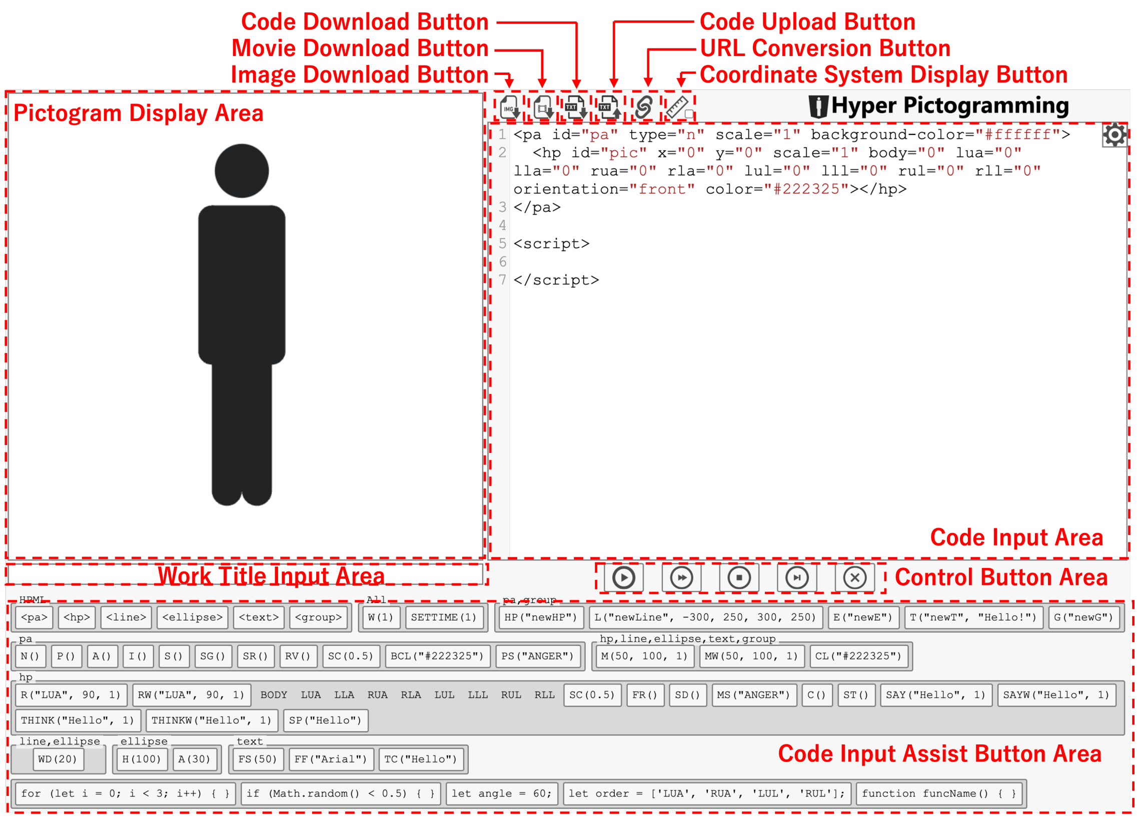
Section 2.1: Access

Please access Hyper Pictogramming via the URL below.

```
https://pictogramming.org/apps/hyperpictogramming/
```

Alternatively, you can search for “Pictogramming” using a search engine and access Hyper Pictogramming from the “Start” button on the Pictogramming series website.

Section 2.2: Screen Overview



```
<pa id="pa" type="n" scale="1" background-color="#ffffff">
  <hp id="pic" x="0" y="0" scale="1" body="0" lua="0"
lla="0" rua="0" rla="0" lul="0" lll="0" rul="0" rll="0"
orientation="front" color="#222325"></hp>
</pa>

<script>

</script>
```

The screen is mainly composed of three parts: the Pictogram Display Area (upper left) that shows the result, the Code Input Area (upper right) where you input code, and the Code Input Assist Button Area (bottom) that helps you input code.

In the middle-right of the screen, there are five control buttons.

Image	Name	Description
▶	Run Button	Run the code in the code editor in the pictogram display area.
⏩	Run Button (fast-forward)	Run the code in the code editor in the pictogram display area in fast-forward mode.
■	Pause Button	Pause execution.
⏭	Resume Button	Resume from the paused state.
✕	Clear Button	Clear all code entered in the code editor.

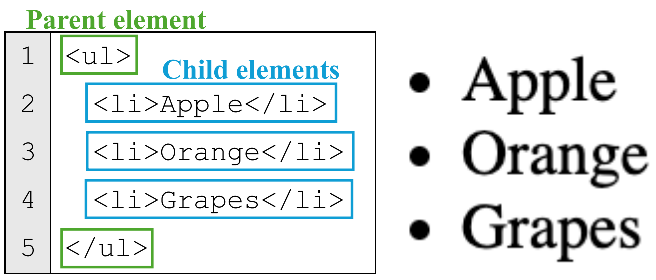
In the upper-right of the screen, there are six buttons for additional functions.

Image	Name	Description
IMG ↓	Image Download Button	Download the pictogram display area as an image.
🎞 ↓	Movie Download Button	Download the pictogram display area as an animation (movie).
TXT ↓	Code Download Button	Download the code in the code editor.
TXT ↑	Code Upload Button	Upload code into the code editor.
🔗	URL Conversion Button	Copy to the clipboard a URL that includes the code editor and the work-title input.
📏	Coordinates System Display Button	Show the coordinate axes in the pictogram display area.

Chapter 3: HPML

Section 3.1: What is HTML?

HTML (HyperText Markup Language) is a markup language that defines the structure of a web page by combining elements. As an example, the elements used to display a list and the resulting output are shown below.



```
<ul>
  <li>Apple</li>
  <li>Orange</li>
  <li>Grapes</li>
</ul>
```

Figure: (Left) HTML example, (Right) output

Elements are written like `'<element>content</element>'`. `'<element>'` is called the start tag, and `'</element>'` is called the end tag. The content of an element can include other elements. In the example above, a `ul` element (unordered list) contains `li` elements (list items). In this case, the `ul` element is the parent element and the `li` elements are child elements.

Besides `ul` and `li`, there are many other elements for different purposes, which can be confusing when you first learn HTML. Therefore, Hyper Pictogramming defines its own markup language called HPML for creating pictograms. Let’s learn HPML.

Section 3.2: HPML (Hyper Pictogram Markup Language)

HPML stands for Hyper Pictogram Markup Language. It is Hyper Pictogramming’s original markup language for creating pictograms. When you access Hyper Pictogramming, the following code is already entered in the Code Input Area. This code displays the initial human pictogram.

```
<pa id="pa" type="n" scale="1" background-color="#ffffff">
  <hp id="pic" x="0" y="0" scale="1" body="0" lua="0" lla="0"
rua="0" rla="0" lul="0" lll="0" rul="0" rll="0"
color="#222325" orientation="front"></hp>
</pa>

<script>

</script>
```

'`id="pa"`', '`id="pic"`', and '`<script> </script>`' are used in Chapter 5. Until then, it is okay to delete them.

This code is composed of `pa` and `hp` elements. `pa` means Pictogram Area, and `hp` means Human Pictogram. We will learn about settings like '`type="n"`' later, so you can ignore them for now.

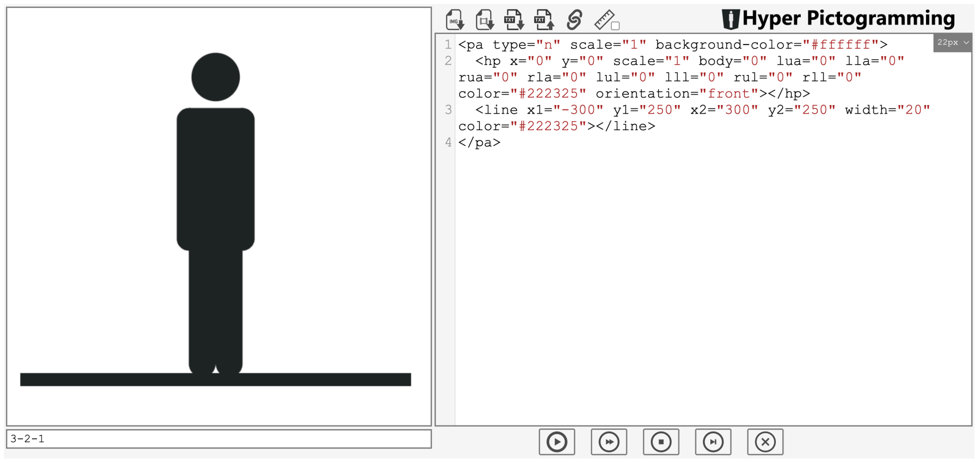
To display a human pictogram, first write a `pa` element as the parent element to create the pictogram area. Then, by writing an `hp` element as a child of the `pa` element, you can display a human pictogram. You cannot display a human pictogram with an `hp` element alone; a `pa` element must be its parent.

Hyper Pictogramming has elements other than `pa` and `hp`, such as `line`, `ellipse`, `text`, and `group`. The `line` element draws a line, the `ellipse` element draws an ellipse, and the `text` element draws text. The `group` element groups child elements together, and you can also nest group elements.

Next, try entering a `line` element under the `hp` element. If you place the cursor under the `hp` element and press the '`<line>`' button in the Code Input Assist Button Area (bottom), you can insert the code easily. After you enter the `line` element, press the Run Button.

Code Example 3-2-1 (←Click to open this code example in Hyper Pictogramming)

```
<pa type="n" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="0"
rla="0" lul="0" lll="0" rul="0" rll="0" color="#222325"
orientation="front"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20"
color="#222325"></line>
</pa>
```



```
<pa type="n" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0"
rua="0" rla="0" lul="0" lll="0" rul="0" rll="0"
color="#222325" orientation="front"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20"
color="#222325"></line>
</pa>
```

You should see a line drawn as in the figure above. However, the human pictogram and the line are still in their default state. Next, let's use attributes to change the pictogram.

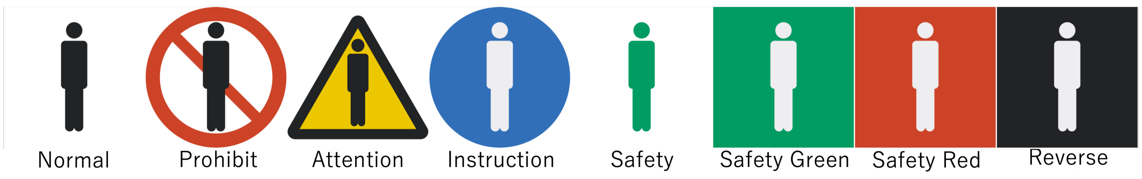
Table: List of HPML tag elements

Tag name	Behavior
`pa`	Create a Pictogram Area.
`hp`	Display a Human Pictogram by making it a child of a `pa` element or a `group` element.
`line`	Draw a line by making it a child of a `pa` element or a `group` element.
`ellipse`	Draw an ellipse by making it a child of a `pa` element or a `group` element.
`text`	Draw text by making it a child of a `pa` element or a group element.
`group`	Make it a child of a `pa` element or a `group` element and use it to group `hp`, `line`, `ellipse`, `text`, and `group` child elements (groups can be nested).

Section 3.3: Attributes

An attribute is written inside a start tag as `'name="value"'`. For example, `pa` has `'type="n"'` and `hp` has `'x="0"'`. By changing attribute values, you can change the appearance of a pictogram.

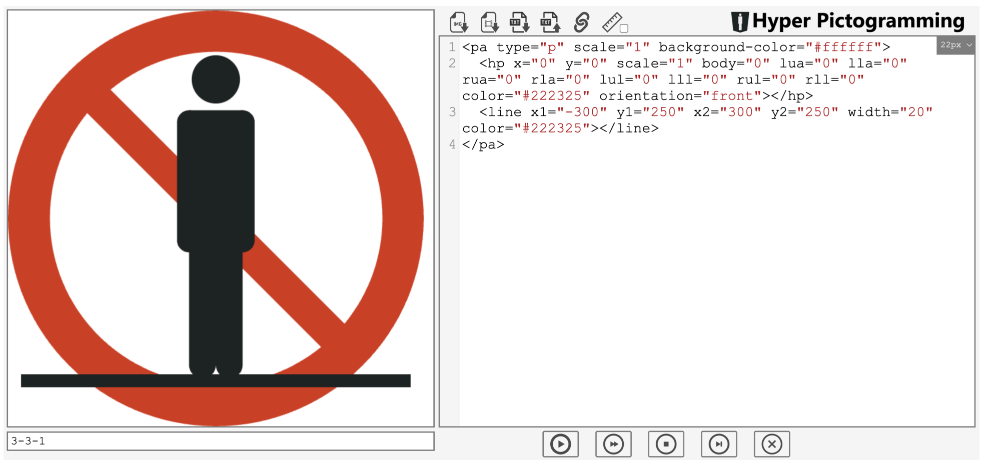
First, look at the `type` attribute of the `pa` element. In the initial state it is `'type="n"'`. `'n'` is short for Normal (standard) and sets the type of the pictogram area.



Try changing the value of the `type` attribute to `'p'`.

Code Example 3-3-1

```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="0" rla="0" lul="0" lll="0" rul="0" rll="0" color="#222325" orientation="front"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20" color="#222325"></line>
</pa>
```



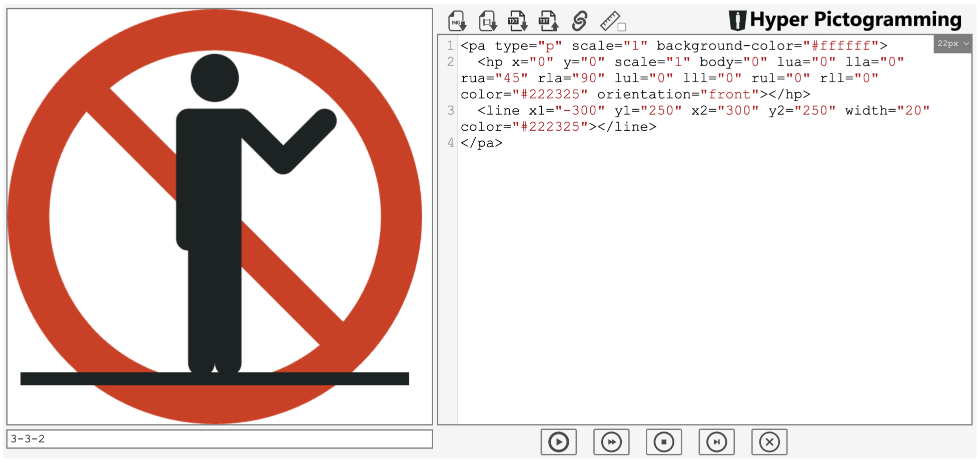
```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="0" rla="0" lul="0" lll="0" rul="0" rll="0" color="#222325" orientation="front"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20" color="#222325"></line>
</pa>
```

You should see that the pictogram becomes a prohibition pictogram as in the figure above. By changing the `pa` element's `type` attribute, you changed the pictogram area to a prohibition style. However, the human pictogram still looks like the initial pose, so let's adjust it next.

First, make the human pictogram look like it is walking while using a smartphone. Create the arm holding the phone by setting `rua` and `rla` to `'45'` and `'90'`, respectively.

Code Example 3-3-2

```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="45" rla="90" lul="0" lll="0" rul="0" rll="0" color="#222325" orientation="front"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20" color="#222325"></line>
</pa>
```



```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0"
rua="45" rla="90" lul="0" lll="0" rul="0" rll="0"
color="#222325" orientation="front"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20"
color="#222325"></line>
</pa>
```

You should see the angles of the right shoulder and right elbow change as in the figure above. `rua` means Right Upper Arm (right shoulder), and `rla` means Right Lower Arm (right elbow).

A list of human pictogram parts is shown below. The body is `body`. Arms and legs are expressed with three letters: the first letter is Left or Right, the second is Upper or Lower, and the third is Arm or Leg.

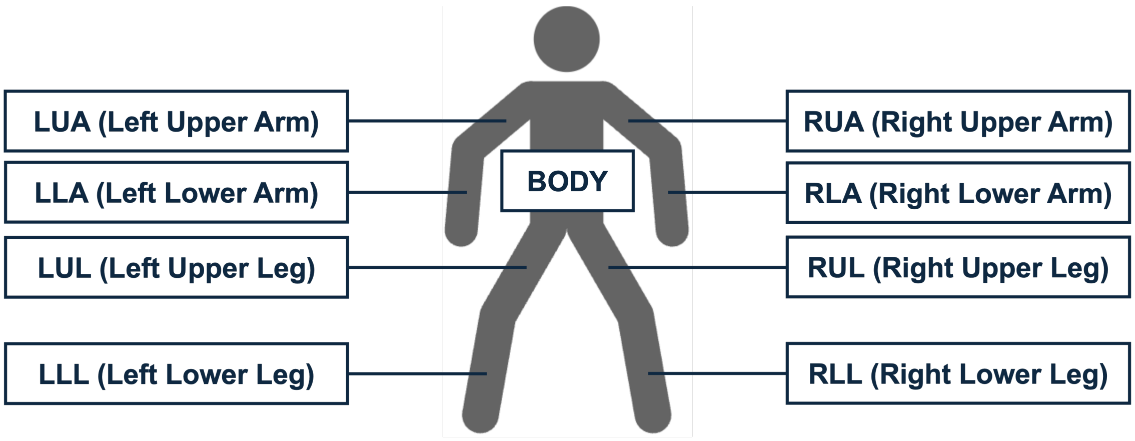
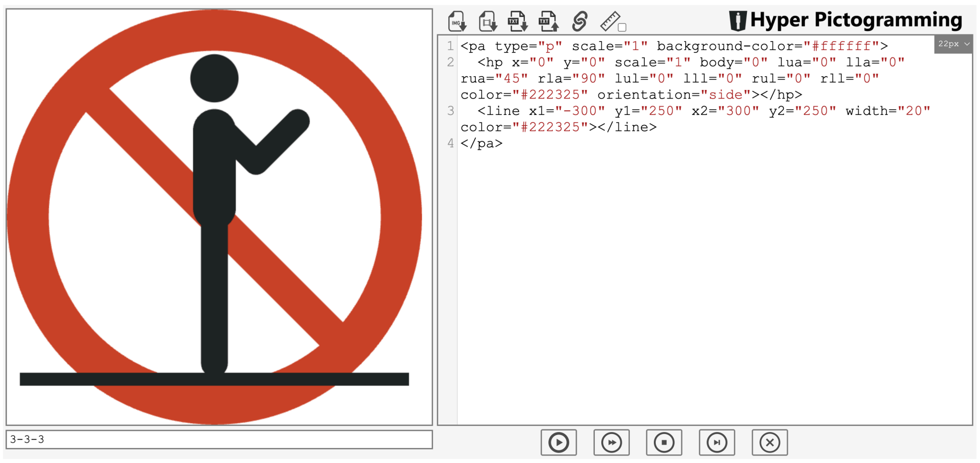


Figure: Parts of a human pictogram

Next, change the orientation attribute, which controls the direction of the human pictogram. Change it from `'front'` (facing forward) to `'side'` (facing sideways).

Code Example 3-3-3

```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="45" rla="90" lul="0" lll="0" rul="0" rll="0" color="#222325" orientation="side"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20" color="#222325"></line>
</pa>
```

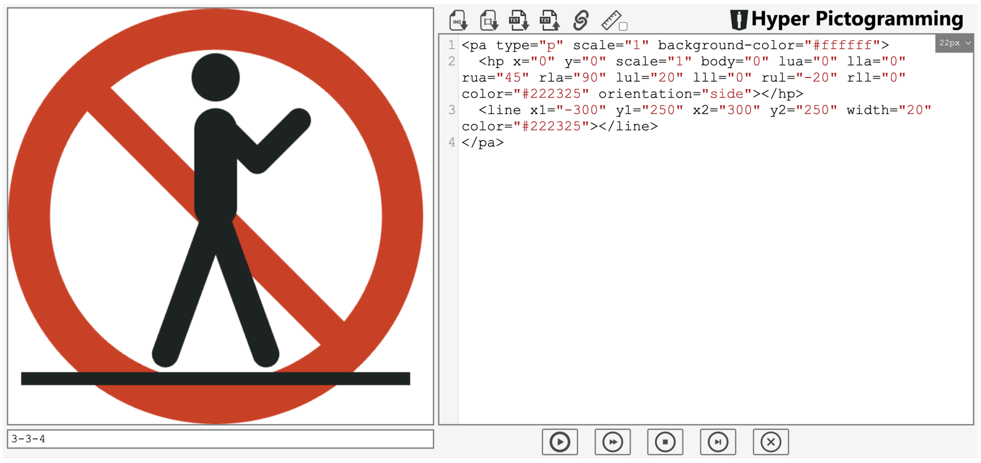


```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="45" rla="90" lul="0" lll="0" rul="0" rll="0" color="#222325" orientation="side"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20" color="#222325"></line>
</pa>
```

You should see the human pictogram facing sideways as in the figure above. Finally, change the angles of the left and right upper legs (`LUL` and `RUL`) so it looks like the person is walking.

Code Example 3-3-4

```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="45" rla="90" lul="20" lll="0" rul="-20" rll="0" color="#222325" orientation="side"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20" color="#222325"></line>
</pa>
```



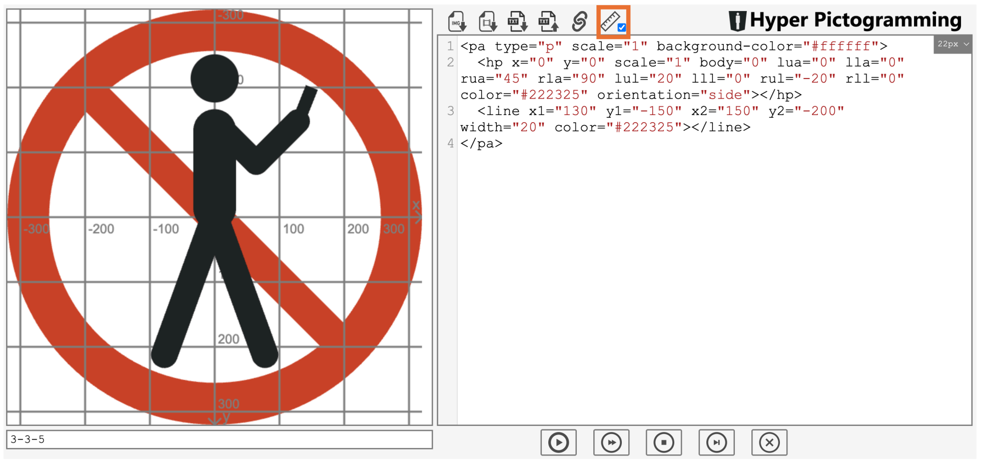
```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0"
rua="45" rla="90" lul="20" lll="0" rul="-20" rll="0"
color="#222325" orientation="side"></hp>
  <line x1="-300" y1="250" x2="300" y2="250" width="20"
color="#222325"></line>
</pa>
```

You should now have been able to create a human pictogram of someone walking while using a smartphone, as in the figure above.

Next, change the attributes of the `line` element to create a smartphone. By changing `x1`, `y1`, `x2`, and `y2`, you can change where the line is drawn. It is helpful to click the Coordinate System Display Button in the upper-right corner of the screen to show the coordinate system.

Code Example 3-3-5

```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0" rua="45"
rla="90" lul="20" lll="0" rul="-20" rll="0" color="#222325"
orientation="side"></hp>
  <line x1="130" y1="-150" x2="150" y2="-200" width="20"
color="#222325"></line>
</pa>
```



```
<pa type="p" scale="1" background-color="#ffffff">
  <hp x="0" y="0" scale="1" body="0" lua="0" lla="0"
rua="45" rla="90" lul="20" lll="0" rul="-20" rll="0"
color="#222325" orientation="side"></hp>
  <line x1="130" y1="-150" x2="150" y2="-200"
width="20" color="#222325"></line>
</pa>
```

You should now be able to create the smartphone as in the figure above. At this point, you have created the “No walking while using a smartphone” pictogram using `pa`, `hp`, and `line` elements.

Below are the attribute lists for each element. Refer to them when needed. If an attribute value remains at its default, you can omit that attribute and the behavior will be the same.

For the `color` and `background-color` attributes, you can specify a color using a color keyword, a hexadecimal string, an RGB function, etc. For example, to specify red, you can use `'red'` as a color keyword, `'#ff0000'` as a hexadecimal value, or `'rgb(255, 0, 0)'` as an RGB function. See MDN documentation for details.

For the `font-family` attribute of the text element, you can specify a font name such as `'Arial'`. See MDN documentation for details.

Table: Attributes of the `pa` element

Attribute	Allowed values	Behavior	Default
`type="arg1"`	`"n"`, `"p"`, `"a"`, `"i"`, `"s"`, `"sg"`, `"sr"`, `"rv"`	Set the type. When `arg1` is `"n"`: Normal, `"p"`: Prohibit, `"a"`: Attention, `"i"`: Instruction, `"s"`: Safety, `"sg"`: Safety Green, `"sr"`: Safety Red, `"rv"`: Reverse.	`"n"`
`scale="arg1"`	Positive number	Set both width and height to `arg1 × 640` pixels.	`1`
`background-color="arg1"`	Color keywords, hexadecimal strings, RGB functions, etc.	Set the background color to `arg1`. Effective only when `type` is `"n"`, `"p"`, `"a"`, `"i"`, or `"s"`; not effective when `type` is `"sg"`, `"sr"`, or `"rv"`.	`"#ffffff"`
`ps="arg1"`	`"none"`, `"anger"`, `"disgust"`, `"fear"`, `"happiness"`, `"sadness"`, `"surprise"`	If `arg1` is `"anger"`, `"disgust"`, `"fear"`, `"happiness"`, `"sadness"`, or `"surprise"`, the corresponding mask is worn. If `arg1` is `"none"`, the mask is removed.	`"none"`

Table: Attributes of the `hp` element

Attribute	Allowed values	Behavior	Default
`x="arg1"`, `y="arg2"`	Number	Set the center coordinate to `(arg1, arg2).`	Both `0`
`scale="arg1"`	Positive number	Scale the size by arg1 relative to the standard (initial) size.	`1`
`body="arg1"`, `lua="arg2"`, `lla="arg3"`, `rua="arg4"`, `rla="arg5"`, `lul="arg6"`, `lll="arg7"`, `rul="arg8"`, `rll="arg9"`	Number	Rotate the Body by `arg1` degrees, Left Upper Arm by `arg2` degrees, Left Lower Arm by `arg3` degrees, Right Upper Arm by `arg4` degrees, Right Lower Arm by `arg5` degrees, Left Upper Leg by `arg6` degrees, Left Lower Leg by `arg7` degrees, Right Upper Leg by `arg8` degrees, Right Lower Leg by `arg9` degrees.	All `0`
`orientation ="arg1"`	`"front"`, or `"side"`	If `arg1` is `"front"`, face forward; if `"side"`, face sideways.	`"front"`
`color="arg1"`	Color keywords, hexadecimal strings, RGB functions, etc.	Set the color to `arg1`.	When the value of the `type` attribute of the `pa` element is `"n"`, `"p"`, or `"a"`, the color is `"#222325"`. When it is `"i"`, `"sg"`, `"sr"`, or `"rv"`, the color is `"#eeedef"`. When it is `"s"`, the color is `"#009c64"`.
`ms="arg1"`	`"none"`, `"anger"`, `"disgust"`, `"fear"`, `"happiness"`, `"sadness"`, `"surprise"`	If `arg1` is `"anger"`, `"disgust"`, `"fear"`, `"happiness"`, `"sadness"`, or `"surprise"`, the corresponding mask is worn. If `arg1` is `"none"`, the mask is removed.	`"none"`
`sk=arg1`	Boolean (`arg1` optional)	Switch the human pictogram to skeleton mode.	`false`

Table: Attributes of the `line` element

Attribute	Allowed values	Behavior	Default
`x1="arg1", y1="arg2", x2="arg3", y2="arg4"`	Number	Set the drawing area from `(arg1, arg2)` to `(arg3, arg4)`.	All `0`
`width="arg1"`	Positive number	Set the thickness to `arg1`.	`20`
`color="arg1"`	Color keywords, hexadecimal strings, RGB functions, etc.	Set the color to `arg1`.	When the value of the `type` attribute of the `pa` element is `"n"`, `"p"`, or `"a"`, the color is `"#222325"`. When it is `"i"`, `"sg"`, `"sr"`, or `"rv"`, the color is `"#eeedef"`. When it is `"s"`, the color is `"#009c64"`.

Table: Attributes of the `ellipse` element

Attribute	Allowed values	Behavior	Default
`x="arg1", y="arg2"`	Number	Set the center coordinate to `(arg1, arg2)`.	Both `0`
`width="arg1", height="arg2"`	Positive number	Set width to `arg1` and height to `arg2`.	Both `100`
`angle="arg1"`	Number	Rotate counterclockwise by `arg1` degrees.	`0`
`color="arg1"`	Color keywords, hexadecimal strings, RGB functions, etc.	Set the color to arg1.	When the value of the `type` attribute of the `pa` element is `"n"`, `"p"`, or `"a"`, the color is `"#222325"`. When it is `"i"`, `"sg"`, `"sr"`, or `"rv"`, the color is `"#eeedef"`. When it is `"s"`, the color is `"#009c64"`.

Table: Attributes of the `text` element

Attribute	Allowed values	Behavior	Default
`x="arg1"`, `y="arg2"`	Number	Set `(arg1, arg2)` as the bottom-left point of the drawing.	Both `0`
`font-size="arg1"`	Positive number	Set the font size to `arg1`.	`50`
`font-family="arg1"`	Font family	Set the font family to `arg1`.	-
`color="arg1"`	Color keywords, hexadecimal strings, RGB functions, etc.	Set the color to `arg1`.	When the value of the `type` attribute of the `pa` element is `"n"`, `"p"`, or `"a"`, the color is `"#222325"`. When it is `"i"`, `"sg"`, `"sr"`, or `"rv"`, the color is `"#eeedef"`. When it is `"s"`, the color is `"#009c64"`.

Table: Attributes of the `group` element

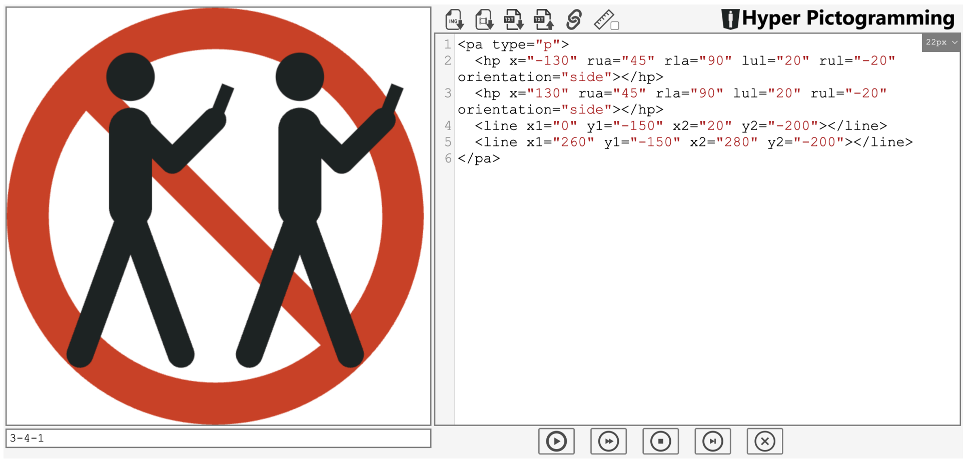
Attribute	Allowed values	Behavior	Default
`x="arg1"`, `y="arg2"`	Number	Set the center coordinate to `(arg1, arg2)`.	Both `0`
`color="arg1"`	Color keywords, hexadecimal strings, RGB functions, etc.	Set the color of all child elements to `arg1`.	When the value of the `type` attribute of the `pa` element is `"n"`, `"p"`, or `"a"`, the color is `"#222325"`. When it is `"i"`, `"sg"`, `"sr"`, or `"rv"`, the color is `"#eeedef"`. When it is `"s"`, the color is `"#009c64"`.

Section 3.4: Adding Elements

You can add multiple elements inside a `pa` element. Add another `hp` element and another `line` element to make two people walking while using smartphones. Adjust positions so they do not overlap.

Code Example 3-4-1

```
<pa type="p">
  <hp x="-130" rua="45" rla="90" lul="20" rul="-20" orientation="side"></hp>
  <hp x="130" rua="45" rla="90" lul="20" rul="-20" orientation="side"></hp>
  <line x1="0" y1="-150" x2="20" y2="-200"></line>
  <line x1="260" y1="-150" x2="280" y2="-200"></line>
</pa>
```



```
<pa type="p">
  <hp x="-130" rua="45" rla="90" lul="20" rul="-20"
orientation="side"></hp>
  <hp x="130" rua="45" rla="90" lul="20" rul="-20"
orientation="side"></hp>
  <line x1="0" y1="-150" x2="20" y2="-200"></line>
  <line x1="260" y1="-150" x2="280" y2="-200"></line>
</pa>
```

You should see two people walking while using smartphones as in the figure above. You can also add multiple `line`, `ellipse`, `text`, and `group` elements as needed.

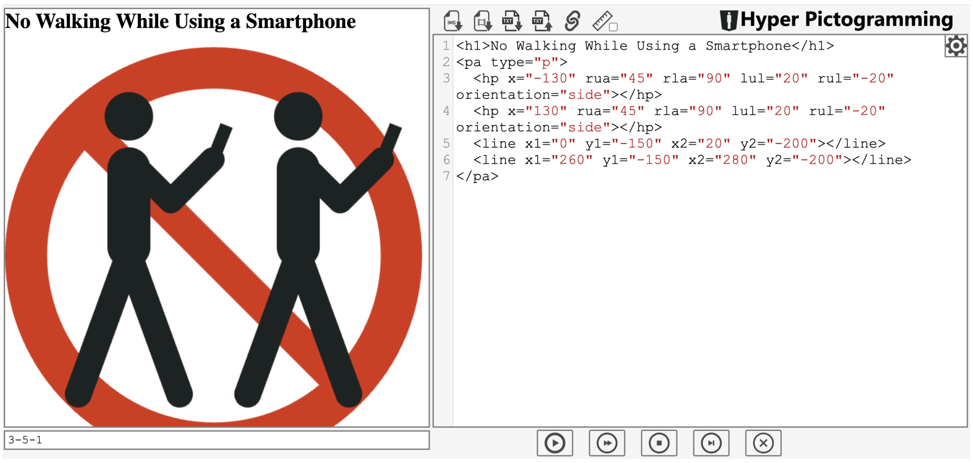
Section 3.5: HPML and HTML

So far, we have used HPML, Hyper Pictogramming's original markup language, to create pictograms. However, Hyper Pictogramming also supports normal HTML, so you can combine HPML with HTML.

Let's combine an `h1` element (a heading) with the "No walking while using a smartphone" pictogram.

Code Example 3-5-1

```
<h1>No Walking While Using a Smartphone</h1>
<pa type="p">
  <hp x="-130" rua="45" rla="90" lul="20" rul="-20" orientation="side"></hp>
  <hp x="130" rua="45" rla="90" lul="20" rul="-20" orientation="side"></hp>
  <line x1="0" y1="-150" x2="20" y2="-200"></line>
  <line x1="260" y1="-150" x2="280" y2="-200"></line>
</pa>
```



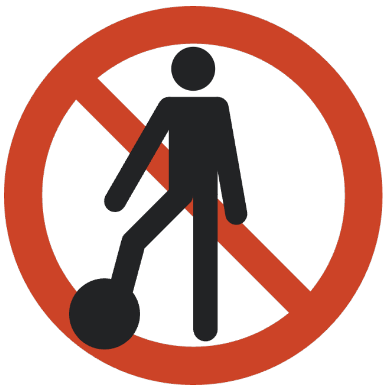
```
<h1>No Walking While Using a Smartphone</h1>
<pa type="p">
  <hp x="-130" rua="45" rla="90" lul="20" rul="-20" orientation="side"></hp>
  <hp x="130" rua="45" rla="90" lul="20" rul="-20" orientation="side"></hp>
  <line x1="0" y1="-150" x2="20" y2="-200"></line>
  <line x1="260" y1="-150" x2="280" y2="-200"></line>
</pa>
```

You should now be able to display both the pictogram and the heading at the same time, as in the figure above.

HTML has many other elements besides `h1`. If you are interested, feel free to try them.

Section 3.6: Exercises

(1) Using HPML, create a pictogram that prohibits playing soccer.

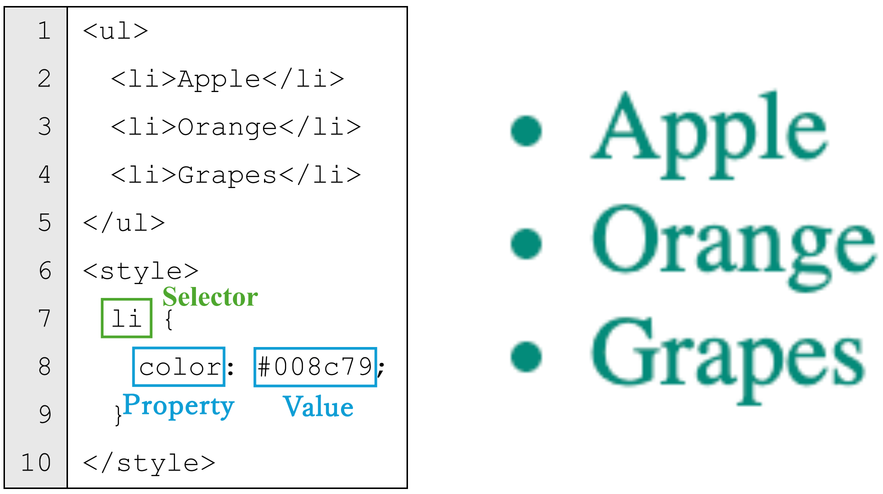


(2) Using HPML, create a pictogram that draws attention / warns about something.

Chapter 4: CSS

Section 4.1: What is CSS?

CSS (Cascading Style Sheets) is used to specify styles for HTML. Below is an example of CSS that makes list text green.



```
<ul>
  <li>Apple</li>
  <li>Orange</li>
  <li>Grapes</li>
</ul>
<style>
  li {
    color: #008c79;
  }
</style>
```

Figure: (Left) CSS example, (Right) output

To apply CSS, first prepare a `style` element (there are other ways such as using a separate CSS file, but in Hyper Pictogramming we use the `style` element).

```
selector {
  property: value;
}
```

A selector specifies which HPML/HTML elements to apply CSS to. In the previous example, `li` is the selector. A property is the type of style, and the property value specifies the value. Although CSS has many properties, Hyper Pictogramming provides properties that are specific to HPML to enable users to learn CSS in a short time. HPML properties and property values correspond to HPML attribute names and attribute values, and they are processed in the same way.

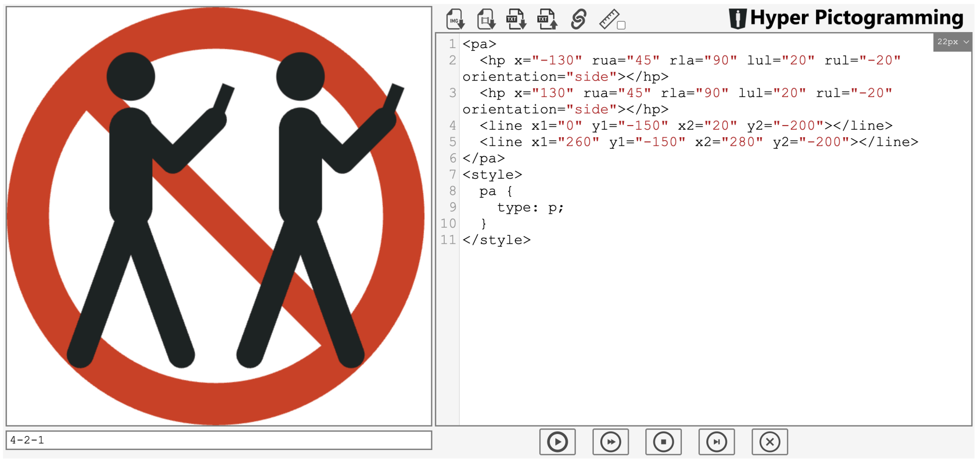
Section 4.2: Applying CSS to a `pa` Element

Now let's use CSS with the `pa` element. First, remove `'type="p"'` from the `pa` element, since

CSS cannot be applied when an HPML attribute is specified. Then, in the `style` element, write the CSS with `pa` as the selector and `type` and `"p"` as the property and value.

Code Example 4-2-1

```
<pa ~~type="p"~~>
  <hp x="-130" rua="45" rla="90" lul="20" rul="-20"
orientation="side"></hp>
  <hp x="130" rua="45" rla="90" lul="20" rul="-20"
orientation="side"></hp>
  <line x1="0" y1="-150" x2="20" y2="-200"></line>
  <line x1="260" y1="-150" x2="280" y2="-200"></line>
</pa>
<style>
  pa {
    type: p;
  }
</style>
```



```
<pa>
  <hp x="-130" rua="45" rla="90" lul="20" rul="-20"
orientation="side"></hp>
  <hp x="130" rua="45" rla="90" lul="20" rul="-20"
orientation="side"></hp>
  <line x1="0" y1="-150" x2="20" y2="-200"></line>
  <line x1="260" y1="-150" x2="280" y2="-200"></line>
</pa>
<style>
  pa {
    type: p;
  }
</style>
```

You should see that you were able to style the `pa` element with CSS as in the figure above.

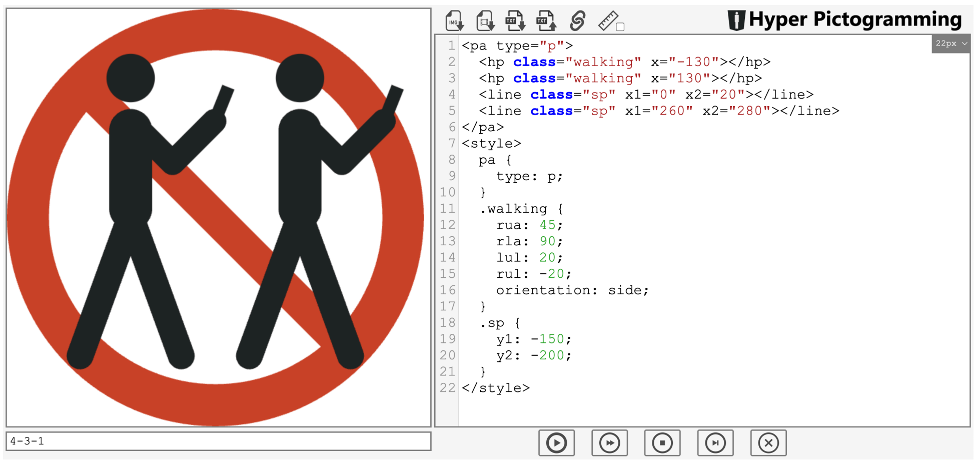
Section 4.3: `class`

Next, apply CSS using the `class` attribute. The `class` attribute lets you apply CSS to multiple elements at once.

Assign the class `'walking'` to the two `hp` elements and `'sp'` to the two `line` elements. Then, to select a class in CSS, prefix the class name with `'.'` (a period).

Code Example 4-3-1

```
<pa type="p">
  <hp class="walking" x="-130" ~~rua="45"~~ ~~rla="90"~~ ~~lul="20"~~ ~~rul="-20"~~ ~~orientation="side"~~></hp>
  <hp class="walking" x="130" ~~rua="45"~~ ~~rla="90"~~ ~~lul="20"~~ ~~rul="-20"~~ ~~orientation="side"~~></hp>
  <line class="sp" x1="0" ~~y1="-150"~~ x2="20" ~~y2="-200"~~></line>
  <line class="sp" x1="260" ~~y1="-150"~~ x2="280" ~~y2="-200"~~></line>
</pa>
<style>
  pa {
    type: p;
  }
  .walking {
    rua: 45;
    rla: 90;
    lul: 20;
    rul: -20;
    orientation: side;
  }
  .sp {
    y1: -150;
    y2: -200;
  }
</style>
```



```
<pa type="p">
  <hp class="walking" x="-130"></hp>
  <hp class="walking" x="130"></hp>
  <line class="sp" x1="0" x2="20"></line>
  <line class="sp" x1="260" x2="280"></line>
</pa>
<style>
  pa {
    type: p;
  }
  .walking {
    rua: 45;
    rla: 90;
    lul: 20;
    rul: -20;
    orientation: side;
  }
  .sp {
    y1: -150;
    y2: -200;
  }
</style>
```

You should see that you can apply CSS using class attributes as in the figure above. Using class attributes allows you to apply CSS with shorter code.

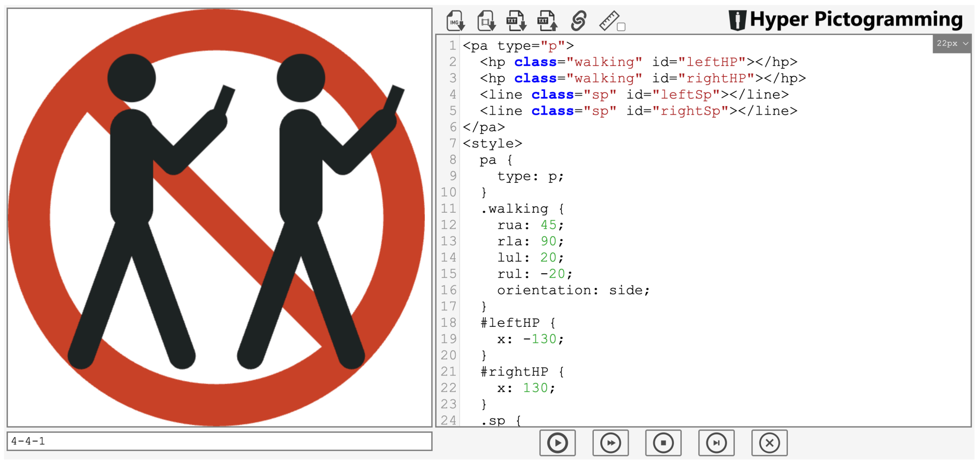
Section 4.4: `id`

Next, apply CSS using the `id` attribute. The `id` attribute uniquely identifies a specific element. Unlike class, the same `id` value must not be used on multiple elements.

Set the first `hp` element's `id` to `'leftHP'`, the second `hp` element's `id` to `'rightHP'`, the first line element's `id` to `'leftSp'`, and the second `line` element's `id` to `'rightSp'`. Then, select them in CSS using `'#'`.

Code Example 4-4-1

```
<pa type="p">
  <hp class="walking" id="leftHP" ~~x="-130"~~></hp>
  <hp class="walking" id="rightHP" ~~x="130"~~></hp>
  <line class="sp" id="leftSp" ~~x1="0"~~ ~~x2="20"~~></line>
  <line class="sp" id="rightSp" ~~x1="260"~~ ~~x2="280"~~></line>
</pa>
<style>
  pa {
    type: p;
  }
  .walking {
    rua: 45;
    rla: 90;
    lul: 20;
    rul: -20;
    orientation: side;
  }
  #leftHP {
    x: -130;
  }
  #rightHP {
    x: 130;
  }
  .sp {
    y1: -150;
    y2: -200;
  }
  #leftSp {
    x1: 0;
    x2: 20;
  }
  #rightSp {
    x1: 260;
    x2: 280;
  }
</style>
```

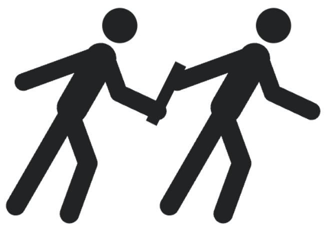


```
<pa type="p">
  <hp class="walking" id="leftHP"></hp>
  <hp class="walking" id="rightHP"></hp>
  <line class="sp" id="leftSp"></line>
  <line class="sp" id="rightSp"></line>
</pa>
<style>
  pa {
    type: p;
  }
  .walking {
    rua: 45;
    rla: 90;
    lul: 20;
    rul: -20;
    orientation: side;
  }
  #leftHP {
    x: -130;
  }
  #rightHP {
    x: 130;
  }
  .sp {
```

You should now have been able to create the “No walking while using a smartphone” pictogram using CSS with `id` and `class` attributes, as in the figure above.

Section 4.5: Exercises

(1) Using HPML and CSS, create a pictogram of a baton pass.



(2) Using HPML and CSS, create a pictogram that gives some instruction.

Chapter 5: JavaScript

Section 5.1: What is JavaScript?

From this chapter, we will create pictograms using JavaScript. JavaScript is a programming language used, for example, to run actions when a user clicks a button on a web page.

When you start learning programming, you may feel it is difficult. In Hyper Pictogramming, you can learn JavaScript in an enjoyable way by manipulating pictograms.

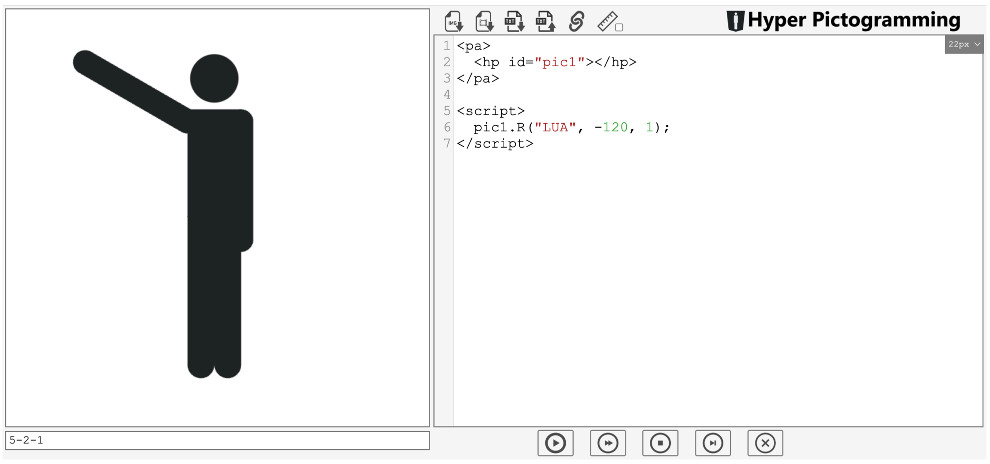
Section 5.2: Sequential and Parallel Execution

To run JavaScript in Hyper Pictogramming, first prepare a `script` element (there are other approaches like using a separate JavaScript file, but in Hyper Pictogramming we use the `script` element), and write your code inside it.

From here, we will learn JavaScript by creating a "waving goodbye" pictogram. First, set the id attribute of the hp element to `'pic1'`. Then, inside the `script` element, write `'pic1.R("LUA", -120, 1);'`.

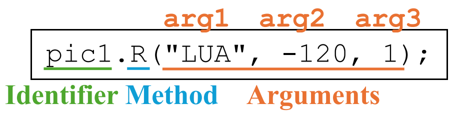
Code Example 5-2-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.R("LUA", -120, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.R("LUA", -120, 1);
</script>
```

You should see the left shoulder of the human pictogram rotate. `'pic1.R("LUA", -120, 1);'` means "`pic1` rotates `LUA` (Left Upper Arm) by `-120` degrees counterclockwise over `1` second." Let's break it down from left to right.



```
pic1.R("LUA", -120, 1);
```

`pic1` is an identifier (name) for the human pictogram. It is the same as the value of the HPML element's `id` attribute. (In Hyper Pictogramming, avoid using "-" in `id` values.)

To run a process on an identifier, write `'.'` (a period). After that comes a method name. A method is an operation you can execute on an identifier.

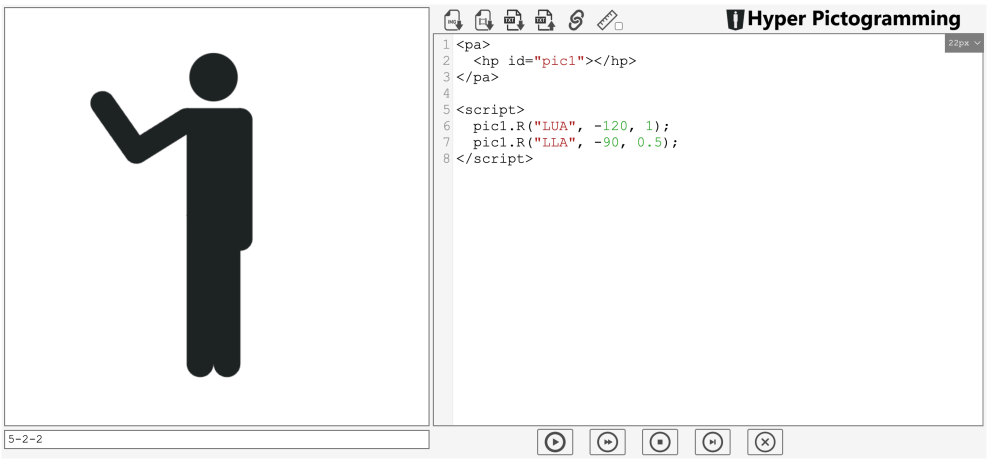
The `R` method means "rotate the body part specified by `arg1` counterclockwise by `arg2` degrees over `arg3` seconds." Therefore, this statement means "`pic1` rotates `LUA` (Left Upper Arm) by –`120` degrees counterclockwise over `1` second." In JavaScript, it is recommended to end statements with `';'` (a semicolon), so a semicolon is added at the end.

Method names and parts are case-insensitive. In other words, `'pic1.R("LUA", -120, 1);'` and `'pic1.r("lua", -120, 1);'` are treated the same.

Next, you want to rotate `LLA` (Left Lower Arm) after `LUA` (Left Upper Arm) finishes rotating. Add the statement `'pic1.R("LLA", -90, 0.5);'` after the previous line.

Code Example 5-2-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.R("LUA", -120, 1);
  pic1.R("LLA", -90, 0.5);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.R("LUA", -120, 1);
  pic1.R("LLA", -90, 0.5);
</script>
```

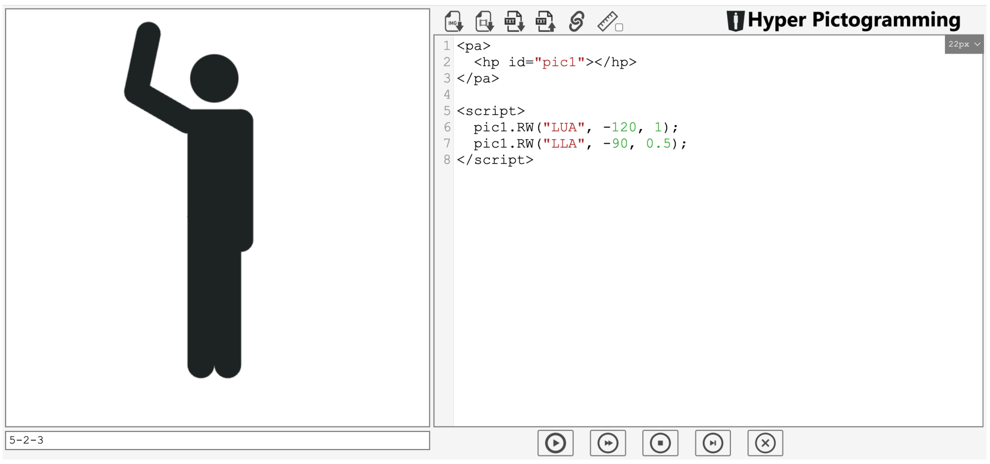
However, you will notice that `LLA` rotates at the same time instead of after `LUA` finishes. This is because the `R` method runs in parallel with subsequent statements.

To rotate `LLA` after `LUA` finishes, use the sequential method (one-by-one execution) instead of the parallel `R` method.

Change the previous two statements from the `R` method to the `RW` method.

Code Example 5-2-3

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.RW("LUA", -120, 1);
  pic1.RW("LLA", -90, 0.5);
</script>
```



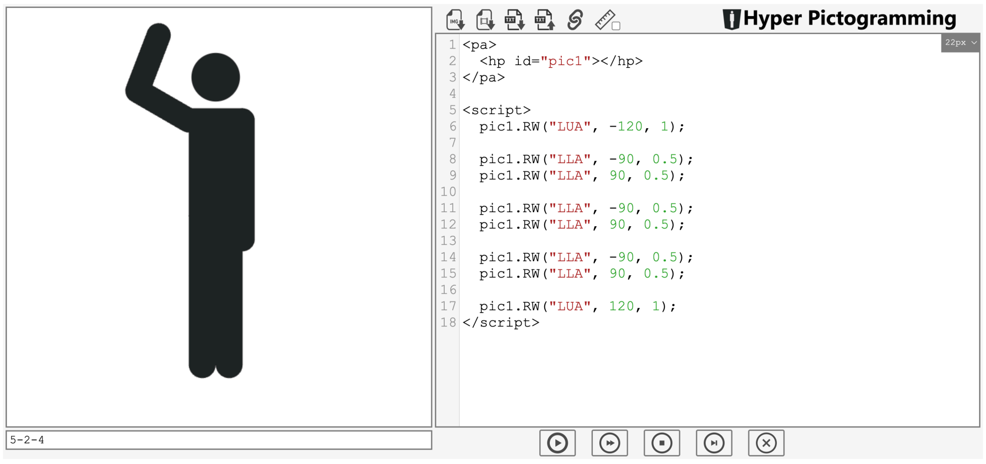
```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.RW("LUA", -120, 1);
  pic1.RW("LLA", -90, 0.5);
</script>
```

You should now be able to rotate LLA after LUA finishes, as intended.

Next, add more RW statements so it looks like waving goodbye. Rotate LLA and then return it, repeating this movement.

Code Example 5-2-4

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.RW("LUA", -120, 1);

  pic1.RW("LLA", -90, 0.5);
  pic1.RW("LLA", 90, 0.5);

  pic1.RW("LLA", -90, 0.5);
  pic1.RW("LLA", 90, 0.5);

  pic1.RW("LLA", -90, 0.5);
  pic1.RW("LLA", 90, 0.5);

  pic1.RW("LUA", 120, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.RW("LUA", -120, 1);

  pic1.RW("LLA", -90, 0.5);
  pic1.RW("LLA", 90, 0.5);

  pic1.RW("LLA", -90, 0.5);
  pic1.RW("LLA", 90, 0.5);

  pic1.RW("LLA", -90, 0.5);
  pic1.RW("LLA", 90, 0.5);

  pic1.RW("LUA", 120, 1);
</script>
```

You should now be able to create a waving-goodbye pictogram that waves three times.

Hyper Pictogramming provides several methods for each element. A list of methods is shown below. Refer to it as needed. (The identifier is the value of the `id` attribute.)

Table: Methods available for the `pa` (Pictogram Area) element

Method signature	Behavior
`*.N();`	Set the type to Normal.
`*.P();`	Set the type to Prohibit.
`*.A();`	Set the type to Attention.
`*.I();`	Set the type to Instruction.
`*.S();`	Set the type to Safety.
`*.SG();`	Set the type to Safety Green.
`*.SR();`	Set the type to Safety Red.
`*.RV();`	Set the type to Reverse.
`*.SC(arg1);`	Set both width and height to `arg1 × 640` pixels.
`*.BCL(arg1);`	Set the background color to `arg1`.
`*.PS(arg1);`	If `arg1` is `"anger"`, `"disgust"`, `"fear"`, `"happiness"`, `"sadness"`, or `"surprise"`, the corresponding mask is worn. If `arg1` is `"none"`, the mask is removed.
`*.HP(arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8, arg9, arg10, arg11, arg12, arg13, arg14, arg15, arg16);`	Add an `hp` child element with `id=arg1`, `x=arg2`, `y=arg3`, `scale=arg4`, `body=arg5`, `lua=arg6`, `lla=arg7`, `rua=arg8`, `rla=arg9`, `lul=arg10`, `lll=arg11`, `rul=arg12`, `rll=arg13`, `orientation=arg14`, `color=arg15`, `ms=arg16`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg16` are omitted, the default value of the corresponding attribute is assumed.
`*.L(arg1, arg2, arg3, arg4, arg5, arg6, arg7);`	Add a `line` child element with `id=arg1`, `x1=arg2`, `y1=arg3`, `x2=arg4`, `y2=arg5`, `width=arg6`, `color=arg7`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg7` are omitted, the default value of the corresponding attribute is assumed.
`*.E(arg1, arg2, arg3, arg4, arg5, arg6, arg7);`	Add an `ellipse` child element with `id=arg1`, `x=arg2`, `y=arg3`, `width=arg4`, `height=arg5`, `angle=arg6`, `color=arg7`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg7` are omitted, the default value of the corresponding attribute is assumed.
`*.T(arg1, arg2, arg3, arg4, arg5, arg6, arg7);`	Add a `text` child element with `id=arg1`, text content is `arg2`, `x=arg3`, `y=arg4`, `font-size=arg5`, `font-family=arg6`, `color=arg7`. If `arg1` and `arg2` are omitted, it is treated as an empty string (`""`), and if any of `arg3` through `arg7` are omitted, the default value of the corresponding attribute is assumed.
`*.G(arg1, arg2, arg3, arg4);`	Add a `group` child element with `id=arg1`, `x=arg2`, `y=arg3`, `color=arg4`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg4` are omitted, the default value of the corresponding attribute is assumed.

Table: Methods available for the hp (Human Pictogram) element

Method signature	Behavior
`*.M(arg1, arg2, arg3, arg4);`	After arg4 seconds, move the whole object at constant speed by arg1 horizontally and arg2 vertically over arg3 seconds. If arg4 is omitted, it is treated as 0. If arg3 and arg4 are both omitted, arg3 is treated as 0.
`*.MW(arg1, arg2, arg3);`	Move the whole object at constant speed by arg1 horizontally and arg2 vertically over arg3 seconds. The next method will not run until the move completes. If arg3 is omitted, it is treated as 0.
`*.R(arg1, arg2, arg3, arg4);`	After arg4 seconds, rotate the body part specified by arg1 about its pivot at constant angular speed by arg2 degrees counterclockwise over arg3 seconds. If arg4 is omitted, it is treated as 0. If arg3 and arg4 are both omitted, arg3 is treated as 0.
`*.RW(arg1, arg2, arg3);`	Rotate the body part specified by arg1 about its pivot at constant angular speed by arg2 degrees counterclockwise over arg3 seconds. The next method will not run until the rotation completes. If arg3 is omitted, it is treated as 0.
`*.SC(arg1);`	Scale the size by arg1 relative to the standard (initial) size.
`*.FR();`	Face forward.
`*.SD();`	Face sideways.
`*.CL(arg1);`	Set the color to arg1.
`*.MS(arg1);`	If arg1 is "anger", "disgust", "fear", "happiness", "sadness", or "surprise", the corresponding mask is worn. If arg1 is "none", the mask is removed.
`*.C();`	Reset to the initial state.
`*.ST();`	Duplicate (stamp) the human pictogram as it is at the time this method is executed.
`*.SAY(arg1, arg2, arg3);`	After arg3 seconds, show the value specified by arg1 in a speech bubble for arg2 seconds. If arg3 is omitted, it is treated as 0. If both arg2 and arg3 are omitted, arg2 is treated as 0.
`*.SAYW(arg1, arg2);`	Show the value specified by arg1 in a speech bubble for arg2 seconds. The next method will not run until the bubble display finishes. If arg2 is omitted, it is treated as 0.
`*.THINK(arg1, arg2, arg3);`	After arg3 seconds, show the value specified by arg1 in a thought bubble for arg2 seconds. If arg3 is omitted, it is treated as 0. If both arg2 and arg3 are omitted, arg2 is treated as 0.
`*.THINKW(arg1, arg2);`	Show the value specified by arg1 in a thought bubble for arg2 seconds. The next method will not run until the bubble display finishes. If arg2 is omitted, it is treated as 0.
`*.SP(arg1, arg2);`	Set the language to arg2 and speak the string arg1. If arg2 is omitted, use "ja-JP" when the browser language is Japanese, otherwise use "en-US".

Table: Methods available for the `line` element

Method signature	Behavior
`*.M(arg1, arg2, arg3, arg4);`	After `arg4` seconds, move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. If `arg4` is omitted, it is treated as `0`. If `arg3` and `arg4` are both omitted, `arg3` is treated as `0`.
`*.MW(arg1, arg2, arg3);`	Move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. The next method will not run until the move completes. If `arg3` is omitted, it is treated as `0`.
`*.WD(arg1);`	Set the width to `arg1`.
`*.CL(arg1);`	Set the color to `arg1`.

Table: Methods available for the `ellipse` element

Method signature	Behavior
`*.M(arg1, arg2, arg3, arg4);`	After `arg4` seconds, move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. If `arg4` is omitted, it is treated as `0`. If `arg3` and `arg4` are both omitted, `arg3` is treated as `0`.
`*.MW(arg1, arg2, arg3);`	Move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. The next method will not run until the move completes. If `arg3` is omitted, it is treated as `0`.
`*.WD(arg1);`	Set the width to `arg1`.
`*.H(arg1);`	Set the height to `arg1`.
`*.A(arg1);`	Set the angle to `arg1` degrees.
`*.CL(arg1);`	Set the color to `arg1`.

Table: Methods available for the `text` element

Method signature	Behavior
`*.M(arg1, arg2, arg3, arg4);`	After `arg4` seconds, move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. If `arg4` is omitted, it is treated as `0`. If `arg3` and `arg4` are both omitted, `arg3` is treated as `0`.
`*.MW(arg1, arg2, arg3);`	Move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. The next method will not run until the move completes. If `arg3` is omitted, it is treated as `0`.
`*.FS(arg1);`	Set the font size to `arg1`.
`*.FF(arg1);`	Set the font family to `arg1`.
`*.CL(arg1);`	Set the color to `arg1`.
`*.TC(arg1);`	Set the text content to `arg1`.

Table: Methods available for the `group` element

Method signature	Behavior
`*.M(arg1, arg2, arg3, arg4);`	After `arg4` seconds, move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. If `arg4` is omitted, it is treated as `0`. If `arg3` and `arg4` are both omitted, `arg3` is treated as `0`.
`*.MW(arg1, arg2, arg3);`	Move the whole object at constant speed by `arg1` horizontally and `arg2` vertically over `arg3` seconds. The next method will not run until the move completes. If `arg3` is omitted, it is treated as `0`.
`*.CL(arg1);`	Set the color of all child elements to `arg1`.
`*.HP(arg1, …, arg16);`	Add an `hp` child element with `id=arg1`, `x=arg2`, `y=arg3`, `scale=arg4`, `body=arg5`, `lua=arg6`, `lla=arg7`, `rua=arg8`, `rla=arg9`, `lul=arg10`, `lll=arg11`, `rul=arg12`, `rll=arg13`, `orientation=arg14`, `color=arg15`, `ms=arg16`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg16` are omitted, the default value of the corresponding attribute is assumed.
`*.L(arg1, …, arg7);`	Add a `line` child element with `id=arg1`, `x1=arg2`, `y1=arg3`, `x2=arg4`, `y2=arg5`, `width=arg6`, `color=arg7`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg7` are omitted, the default value of the corresponding attribute is assumed.
`*.E(arg1, …, arg7);`	Add an `ellipse` child element with `id=arg1`, `x=arg2`, `y=arg3`, `width=arg4`, `height=arg5`, `angle=arg6`, `color=arg7`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg7` are omitted, the default value of the corresponding attribute is assumed.
`*.T(arg1, …, arg7);`	Add a `text` child element with `id=arg1`, text content is `arg2`, `x=arg3`, `y=arg4`, `font-size=arg5`, `font-family=arg6`, `color=arg7`. If `arg1` and `arg2` are omitted, it is treated as an empty string (`""`), and if any of `arg3` through `arg7` are omitted, the default value of the corresponding attribute is assumed.
`*.G(arg1, …, arg4);`	Add a `group` child element with `id=arg1`, `x=arg2`, `y=arg3`, `color=arg4`. If `arg1` is omitted, it is treated as an empty string (`""`), and if any of `arg2` through `arg4` are omitted, the default value of the corresponding attribute is assumed.

Table: Methods available for all elements

Method signature	Behavior
`*.W(arg1);`	Wait `arg1` seconds without executing the next method.
`*.SETTIME(arg1);`	Set the current time to `arg1` seconds.
`*.SEND_MESSAGE(arg1, arg2);`	If `arg2` is an object, send message `arg1` to `arg2`; if `arg2` is an array, send it to all objects in `arg2`.
`*.SEND_MESSAGE_TO_ALL(arg1);`	Send message `arg1` to all objects.
`*.RECEIVE_MESSAGE(arg1, arg2);`	When message `arg1` is received, execute `arg2`.

Section 5.3: Variables and Constants

Suppose you want to wave a little faster. To do that, you need to change the third argument of the `RW` method for waving. But since you already wrote many `RW` statements, changing each one is tedious. So we will use constants and variables.

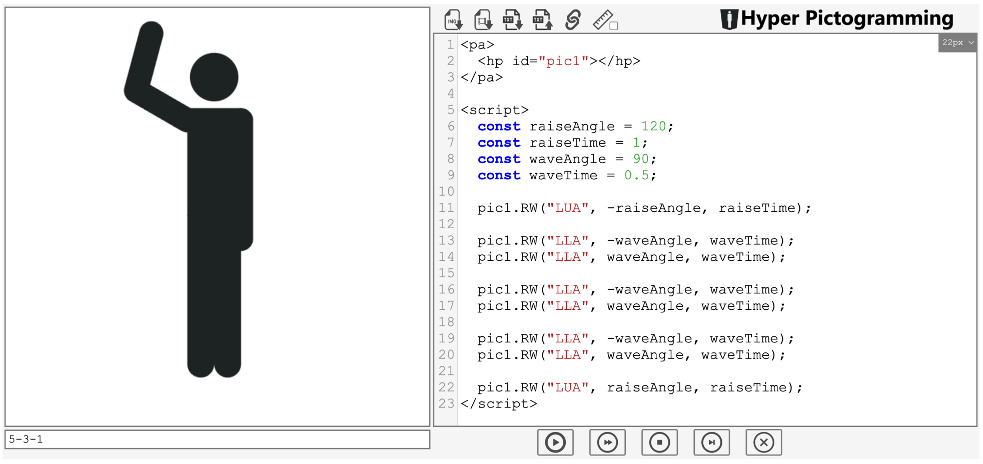
Signature	Behavior
`const arg1 = arg2`	Assign `arg2` to constant `arg1`.
`let arg1 = arg2`	Assign `arg2` to variable `arg1`.

A constant is a value that cannot be changed later, while a variable is a value that can be changed later. You can define constants with const and variables with let.

Define values such as the waving time as constants.

Code Example 5-3-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LUA", raiseAngle, raiseTime);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LUA", raiseAngle, raiseTime);
</script>
```

You should now be able to create the waving pictogram using constants. With constants, if you want to change the waving speed later, you only need to change the values assigned to `waveTime`, etc.

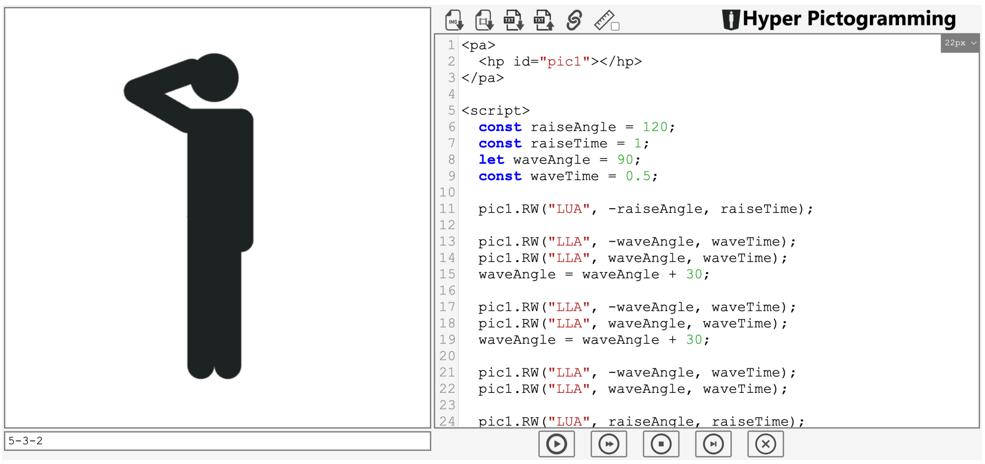
Next, you want the waving angle to gradually become larger. For that, use arithmetic operators. Arithmetic operators are symbols used for calculations.

Signature	Behavior
`A + B`	Add `A` and `B`.
`A - B`	Subtract `B` from `A`.
`A * B`	Multiply `A` and `B`.
`A / B`	Divide `A` by `B`.
`A % B`	Remainder when `A` is divided by `B`.
`A ** B`	Raise `A` to the power of `B`.

Change `waveAngle` from a constant to a variable. Then, use `'+'` to gradually increase the value of `waveAngle`.

Code Example 5-3-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);
  waveAngle = waveAngle + 30;

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);
  waveAngle = waveAngle + 30;

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LUA", raiseAngle, raiseTime);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);
  waveAngle = waveAngle + 30;

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);
  waveAngle = waveAngle + 30;

  pic1.RW("LLA", -waveAngle, waveTime);
  pic1.RW("LLA", waveAngle, waveTime);

  pic1.RW("LUA", raiseAngle, raiseTime);
```

You should see that the waving angle gradually becomes larger.

Section 5.4: Loops

At this point, you have created the waving-goodbye pictogram. However, you are repeating similar statements many times. Waving three times is manageable, but for more repetitions the code becomes long. So, we will use a loop.

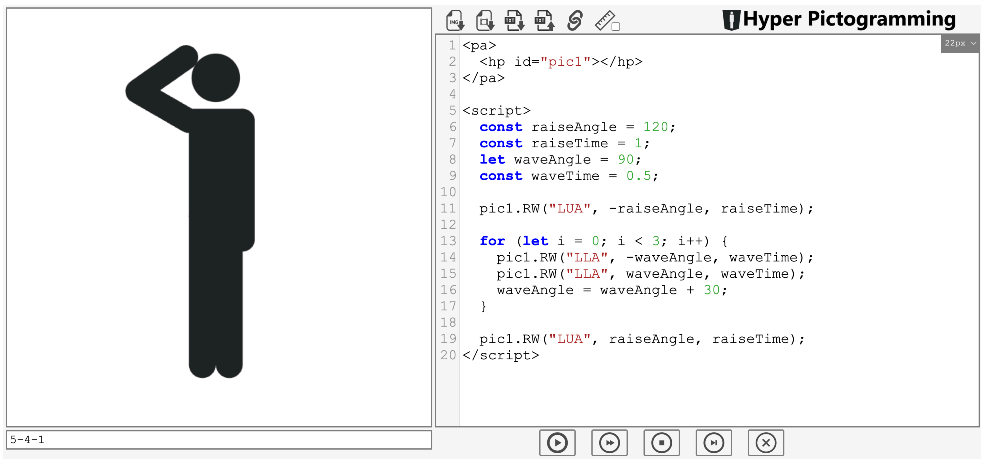
Signature	Behavior
`for (let i = 0; i < arg1; i++) { }`	Repeat the corresponding block `arg1` times.

In a `for` statement, `'let i = 0'` initializes before the loop starts, `'i < arg1'` is evaluated at the start of each iteration, and `'i++'` runs at the end of each iteration.

By using a `for` loop, you can wave goodbye with shorter code as shown below.

Code Example 5-4-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);

  for (let i = 0; i < 3; i++) {
    pic1.RW("LLA", -waveAngle, waveTime);
    pic1.RW("LLA", waveAngle, waveTime);
    waveAngle = waveAngle + 30;
  }

  pic1.RW("LUA", raiseAngle, raiseTime);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);

  for (let i = 0; i < 3; i++) {
    pic1.RW("LLA", -waveAngle, waveTime);
    pic1.RW("LLA", waveAngle, waveTime);
    waveAngle = waveAngle + 30;
  }

  pic1.RW("LUA", raiseAngle, raiseTime);
</script>
```

You should now be able to wave with gradually larger angles using a `for` loop and shorter code.

Section 5.5: Conditional Branch

So far, you have waved with the left hand. Next, you want to wave randomly with either the left or the right hand. For that, use conditional branching with an if statement.

Signature	Behavior
`if (exp1) { }`	If expression `exp1` is true, execute the corresponding statement block.

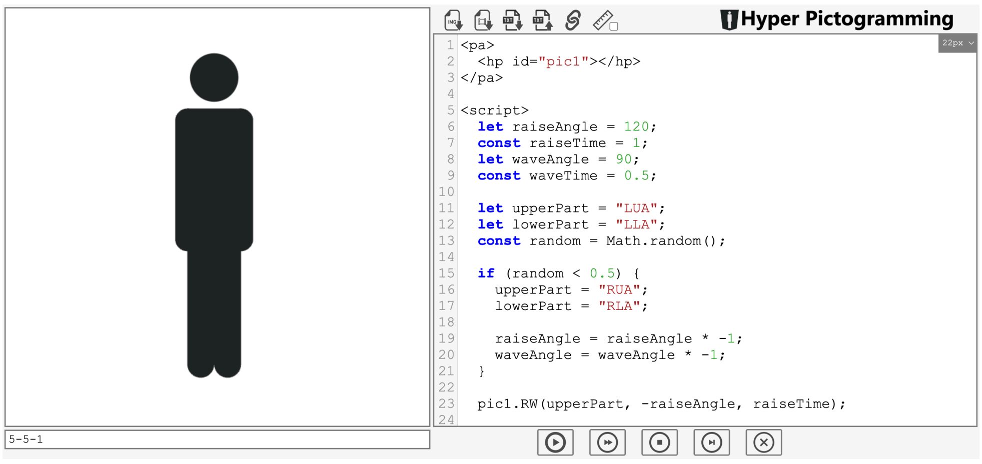
In `exp1`, write a conditional expression using comparison operators.

Signature	Evaluation
`A > B`	`A` is greater than `B`.
`A >= B`	`A` is greater than or equal to `B`.
`A < B`	`A` is less than `B`.
`A <= B`	`A` is less than or equal to `B`.
`A == B`	`A` is equal to `B`.
`A != B`	`A` is not equal to `B`.
`A++, ++A`	Increment `A` by 1 (same as `A=A+1` or `A+=1`).
`A--, --A`	Decrement `A` by 1 (same as `A=A-1` or `A-=1`).

To wave with the left hand 50% of the time and the right hand 50% of the time, use `Math.random()`, which returns a random value from 0 (inclusive) to 1 (exclusive). If the value generated by `Math.random()` is less than 0.5, modify the program so that it waves with the right hand. You do not need to memorize `Math.random()`. When necessary, you can simply search using keywords such as "JavaScript random number."

Code Example 5-5-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  let raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  let upperPart = "LUA";
  let lowerPart = "LLA";
  const random = Math.random();

  if (random < 0.5) {
    upperPart = "RUA";
    lowerPart = "RLA";

    raiseAngle = raiseAngle * -1;
    waveAngle = waveAngle * -1;
  }

  pic1.RW(upperPart, -raiseAngle, raiseTime);

  for (let i = 0; i < 3; i++) {
    pic1.RW(lowerPart, -waveAngle, waveTime);
    pic1.RW(lowerPart, waveAngle, waveTime);
  }

  pic1.RW(upperPart, raiseAngle, raiseTime);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  let raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  let upperPart = "LUA";
  let lowerPart = "LLA";
  const random = Math.random();

  if (random < 0.5) {
    upperPart = "RUA";
    lowerPart = "RLA";

    raiseAngle = raiseAngle * -1;
    waveAngle = waveAngle * -1;
  }

  pic1.RW(upperPart, -raiseAngle, raiseTime);

```

You should now be able to wave with the left or right hand with a 50% probability each.

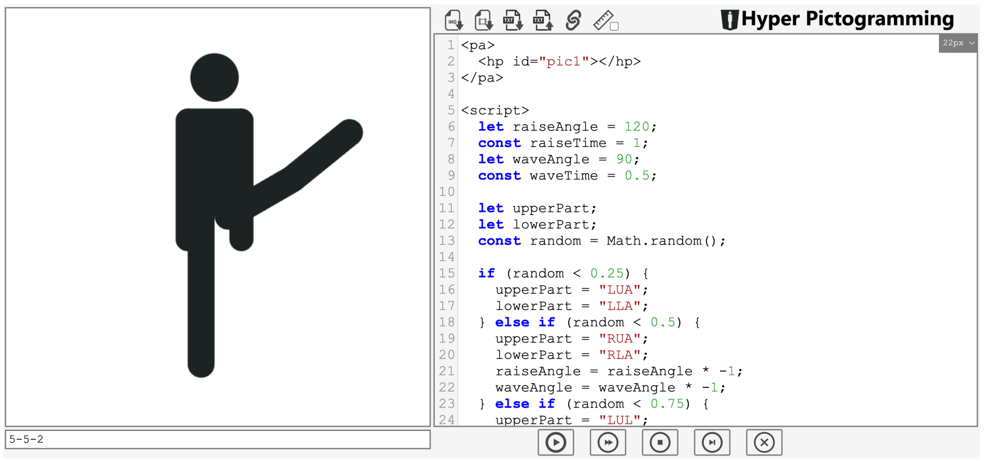
You can also use `else if` and `else` to make more complex branching.

Signature	Behavior
`if (exp1) {}`	If `exp1` is true, execute the corresponding block.
`else if (expN) {}`	If all previous `if/else if` conditions are false and `expN` is true, execute the corresponding block.
`else {}`	If all previous `if/else if` conditions are false, execute the corresponding block.

Make it wave goodbye with the left hand, right hand, left foot, or right foot with 25% probability each.

Code Example 5-5-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  let raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  let upperPart;
  let lowerPart;
  const random = Math.random();

  if (random < 0.25) {
    upperPart = "LUA";
    lowerPart = "LLA";
  } else if (random < 0.5) {
    upperPart = "RUA";
    lowerPart = "RLA";
    raiseAngle = raiseAngle * -1;
    waveAngle = waveAngle * -1;
  } else if (random < 0.75) {
    upperPart = "LUL";
    lowerPart = "LLL";
  } else {
    upperPart = "RUL";
    lowerPart = "RLL";
    raiseAngle = raiseAngle * -1;
    waveAngle = waveAngle * -1;
  }

  pic1.RW(upperPart, -raiseAngle, raiseTime);

  for (let i = 0; i < 3; i++) {
    pic1.RW(lowerPart, -waveAngle, waveTime);
    pic1.RW(lowerPart, waveAngle, waveTime);
  }

  pic1.RW(upperPart, raiseAngle, raiseTime);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  let raiseAngle = 120;
  const raiseTime = 1;
  let waveAngle = 90;
  const waveTime = 0.5;

  let upperPart;
  let lowerPart;
  const random = Math.random();

  if (random < 0.25) {
    upperPart = "LUA";
    lowerPart = "LLA";
  } else if (random < 0.5) {
    upperPart = "RUA";
    lowerPart = "RLA";
    raiseAngle = raiseAngle * -1;
    waveAngle = waveAngle * -1;
  } else if (random < 0.75) {
    upperPart = "LUL";
```

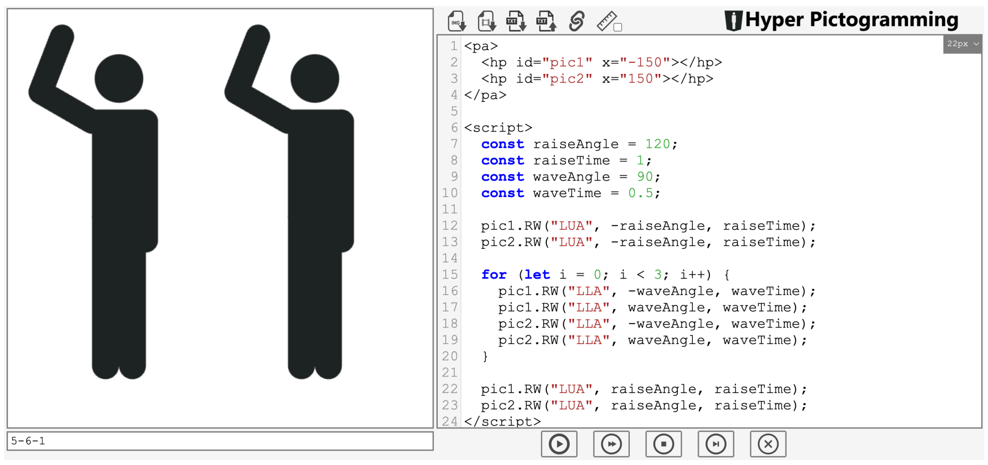
You should now be able to wave with the left hand, right hand, left foot, and right foot with 25% probability each.

Section 5.6: Procedures and Functions

Next, make two people wave goodbye. First, add an `hp` element whose `id` attribute value is `'pic2'`. Adjust the `x` attribute so the two people do not overlap. Then, apply the same code to `pic2` as well.

Code Example 5-6-1

```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);
  pic2.RW("LUA", -raiseAngle, raiseTime);

  for (let i = 0; i < 3; i++) {
    pic1.RW("LLA", -waveAngle, waveTime);
    pic1.RW("LLA", waveAngle, waveTime);
    pic2.RW("LLA", -waveAngle, waveTime);
    pic2.RW("LLA", waveAngle, waveTime);
  }

  pic1.RW("LUA", raiseAngle, raiseTime);
  pic2.RW("LUA", raiseAngle, raiseTime);
</script>
```



```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  pic1.RW("LUA", -raiseAngle, raiseTime);
  pic2.RW("LUA", -raiseAngle, raiseTime);

  for (let i = 0; i < 3; i++) {
    pic1.RW("LLA", -waveAngle, waveTime);
    pic1.RW("LLA", waveAngle, waveTime);
    pic2.RW("LLA", -waveAngle, waveTime);
    pic2.RW("LLA", waveAngle, waveTime);
  }

  pic1.RW("LUA", raiseAngle, raiseTime);
  pic2.RW("LUA", raiseAngle, raiseTime);
</script>
```

You should now be able to wave with two people. Note that waiting (blocking) with the `RW` method affects only that identifier. So even if you use `RW` on `pic1`, `pic2`'s code can run in parallel.

However, if you want to wave with 10 people, the code becomes long and hard to manage. Also, it becomes unclear what the code is intended to do. So we will bundle the behavior into a procedure.

A procedure is reusable code that groups a series of steps and can be called later. It makes code shorter and cleaner, and lets you give a name to a series of operations.

Signature	Behavior
`function name(arg1, …, argN) { }`	Define a procedure name with N arguments `arg1` through `argN`.

Code Example 5-6-2

```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  function byebye() {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 3; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);
  }

  pic1.byebye = byebye;
  pic1.byebye();
  pic2.byebye = byebye;
  pic2.byebye();
</script>
```

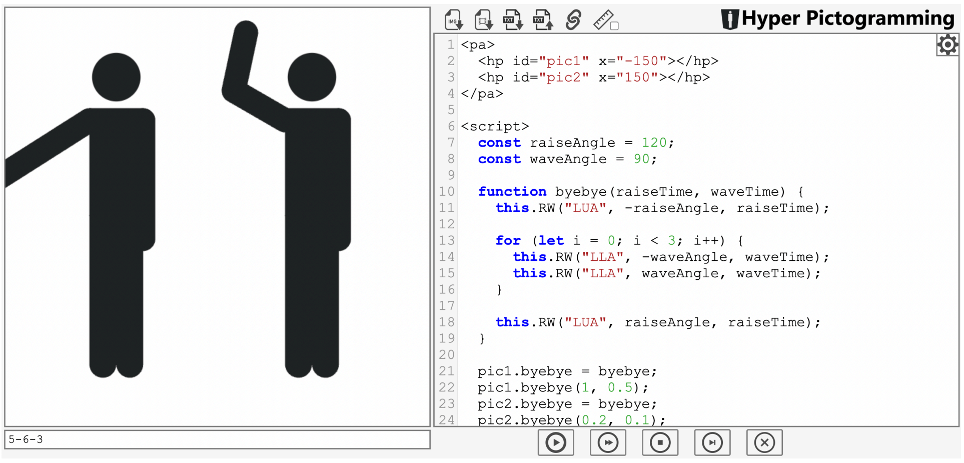
You should now be able to wave with two people as before. `'picN.byebye = byebye'` assigns the procedure `byebye` to `picN` as a method named `byebye`, so you can call it with `'picN.byebye()'`.

By using the `byebye` procedure, you can make multiple people wave with short code. Next, let's add parameters (arguments) to the `byebye` procedure.

Previously, `raiseTime` and `waveTime` were defined as constants. Now make them arguments. When you call the procedure, pass the values for `raiseTime` and `waveTime`.

Code Example 5-6-3

```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  ~~const raiseTime = 1;~~
  const waveAngle = 90;
  ~~const waveTime = 0.5;~~

  function byebye(raiseTime, waveTime) {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 3; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);
  }

  pic1.byebye = byebye;
  pic1.byebye(1, 0.5);
  pic2.byebye = byebye;
  pic2.byebye(0.2, 0.1);
</script>
```



```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const waveAngle = 90;

  function byebye(raiseTime, waveTime) {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 3; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);
  }

  pic1.byebye = byebye;
  pic1.byebye(1, 0.5);
  pic2.byebye = byebye;
  pic2.byebye(0.2, 0.1);
```

You should now be able to make two people wave at different speeds.

So far, we have used function to define a procedure. Note that some functions return a value. When a function returns a value, it is usually called a function rather than a procedure.

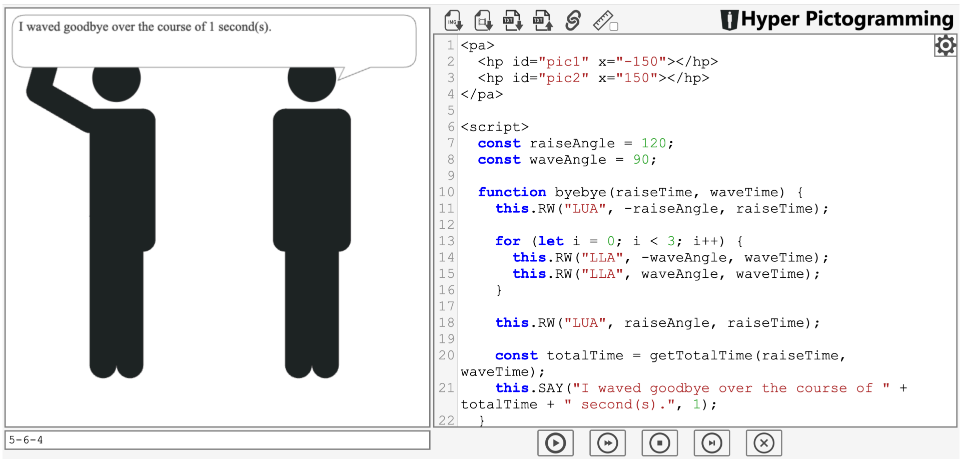
Signature	Behavior

`function name(arg1, …, argN) {`
`  return result;`
`}`
	Define a function `name` with N arguments `arg1` through `argN` that returns `result`.

Define a function `getTotalTime` that returns how many seconds it took to wave goodbye based on `raiseTime` and `waveTime`. Display the returned value using the `SAY` method.

Code Example 5-6-4

```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const waveAngle = 90;

  function byebye(raiseTime, waveTime) {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 3; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);

    const totalTime = getTotalTime(raiseTime, waveTime);
    this.SAY("I waved goodbye over the course of " + totalTime + " second(s).", 1);
  }

  function getTotalTime(raiseTime, waveTime) {
    let totalTime = raiseTime * 2 + waveTime * 3 * 2;
    return totalTime;
  }

  pic1.byebye = byebye;
  pic1.byebye(1, 0.5);
  pic2.byebye = byebye;
  pic2.byebye(0.2, 0.1);
</script>
```



```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  const raiseAngle = 120;
  const waveAngle = 90;

  function byebye(raiseTime, waveTime) {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 3; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);

    const totalTime = getTotalTime(raiseTime,
waveTime);
    this.SAY("I waved goodbye over the course of " +
totalTime + " second(s).", 1);
  }
```

Section 5.7: Arrays

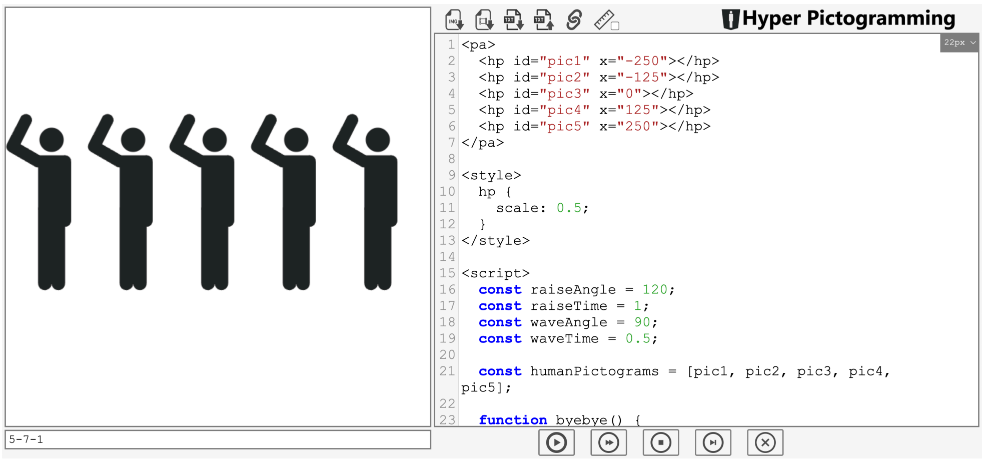
Next, you want five people to wave goodbye. You could repeat `'picN.byebye = byebye; picN.byebye();'`, but it becomes long, so we will use an array.

An array is a data structure for storing multiple values. You can store values separated by `','` (commas) inside `'[]'`. Prepare an array named `humanPictograms` and store identifiers like `pic1`, `pic2`, etc. in it.

To get elements from an array, write something like `humanPictograms[0]`. Note that the first element is index 0 (not 1).

Code Example 5-7-1

```
<pa>
  <hp id="pic1" x="-250"></hp>
  <hp id="pic2" x="-125"></hp>
  <hp id="pic3" x="0"></hp>
  <hp id="pic4" x="125"></hp>
  <hp id="pic5" x="250"></hp>
</pa>

<style>
  hp {
    scale: 0.5;
  }
</style>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  const humanPictograms = [pic1, pic2, pic3, pic4, pic5];

  function byebye() {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 3; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);
  }

  for (let i = 0; i < humanPictograms.length; i++) {
    humanPictograms[i].byebye = byebye;
    humanPictograms[i].byebye();
  }
</script>
```



```
<pa>
  <hp id="pic1" x="-250"></hp>
  <hp id="pic2" x="-125"></hp>
  <hp id="pic3" x="0"></hp>
  <hp id="pic4" x="125"></hp>
  <hp id="pic5" x="250"></hp>
</pa>

<style>
  hp {
    scale: 0.5;
  }
</style>

<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;

  const humanPictograms = [pic1, pic2, pic3, pic4,
pic5];

  function byebye() {
```

By combining arrays and a `for` loop, you can wave goodbye with multiple people using short code.

Section 5.8: Method Chaining

Finally, let’s introduce method chaining. Method chaining is a way to write methods by connecting them one after another.

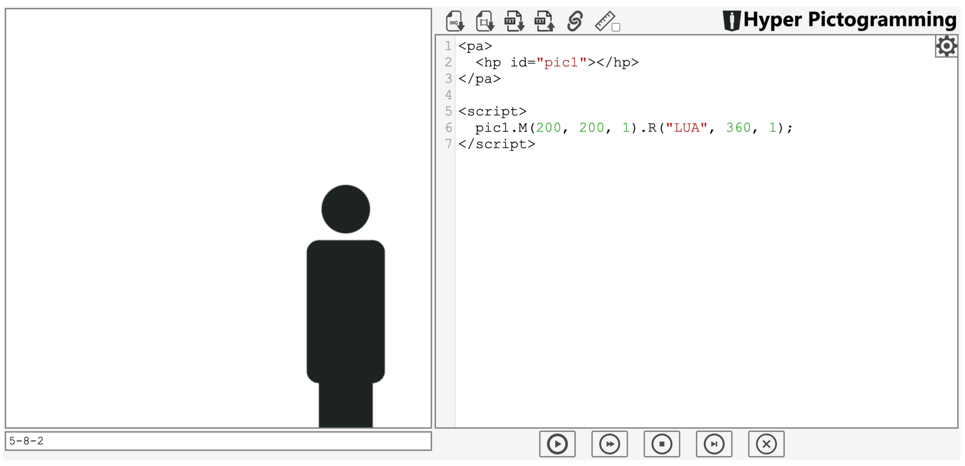
For example, the following two code snippets look different, but produce the same result.

Code Example 5-8-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.M(200, 200, 1);
  pic1.R("LUA", 360, 1);
</script>
```

Code Example 5-8-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.M(200, 200, 1).R("LUA", 360, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.M(200, 200, 1).R("LUA", 360, 1);
</script>
```

Using method chaining can make code more concise and make the flow of processing easier to understand.

Section 5.9: Exercises

(1) Using JavaScript, make the human pictogram(s) in the “No walking while using a smartphone” pictogram you created in Chapter 3 (HPML) and Chapter 4 (CSS) actually walk.

(2) Using HPML and JavaScript, create an animated/moving pictogram.

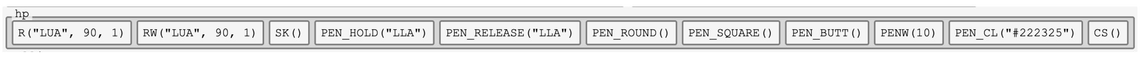
Chapter 6: Pictographics

This chapter covers Pictographics. In Pictographics, a human pictogram holds a pen, and its movement draws the pen’s trail.

Please access the URL below.

```
https://pictogramming.org/apps/hyperpictogramming/?pictographics=true
```

The screen is almost the same as before, but the hp buttons in the Code Input Assist Button Area change. As shown in the image below, buttons corresponding to the methods in this chapter are displayed.



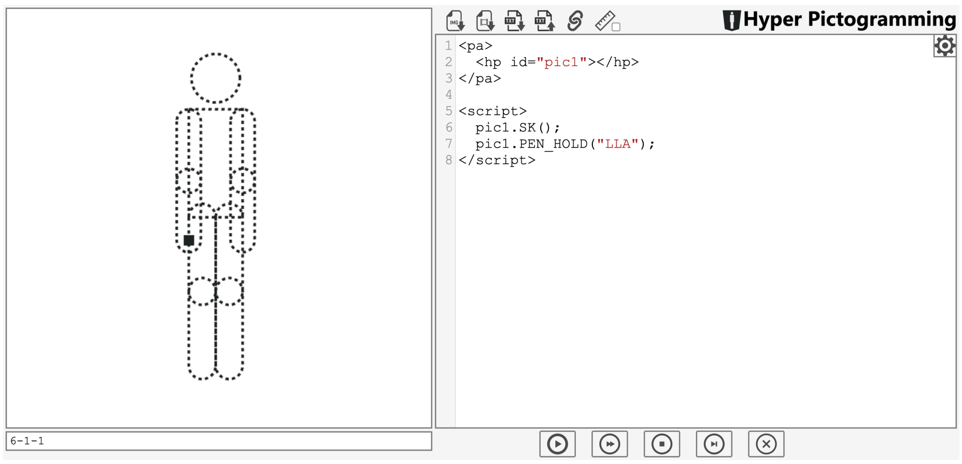
Note: You can use this chapter’s methods on the previous URL as well, and you can also use the methods from the previous URL on this one.

Section 6.1: Holding a Pen

Let’s make the human pictogram hold a pen. First, to make the pen trail easier to see, use the `SK` method to switch the human pictogram to skeleton mode. Then, use the `PEN_HOLD` method.

Code Example 6-1-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");
</script>
```

You were able to hold a pen in the left hand. In the `PEN_HOLD` method, `arg1` specifies a body part. You can specify the same parts that you can with the `R` and `RW` methods.

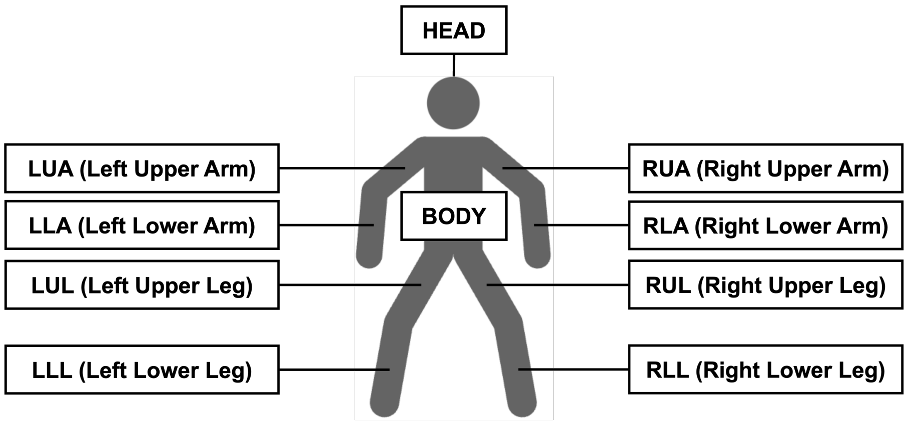


Figure: Parts of a human pictogram

There is also the `PEN_RELEASE` method to release the pen, and the `CS` method to clear what has been drawn so far.

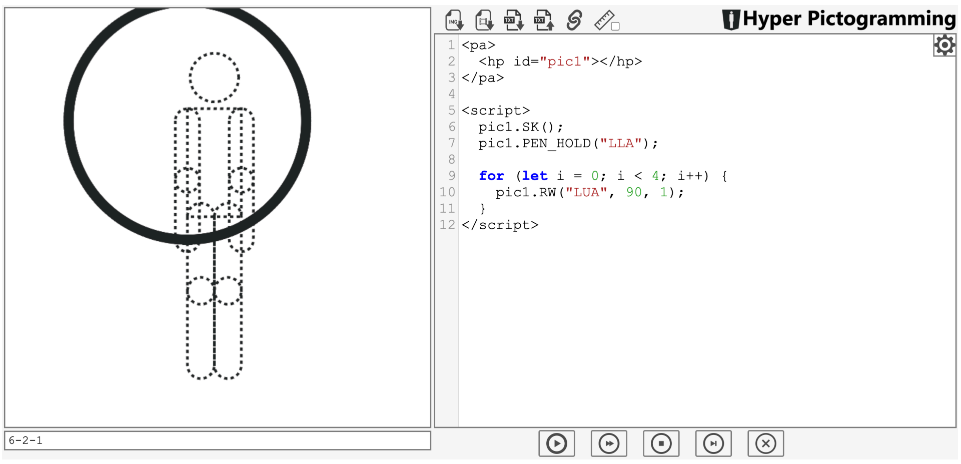
Method signature	Behavior
`*.SK(arg1);`	If `arg1` is omitted, switch the human pictogram to skeleton mode. If `arg1` is `"None"`, switch to normal mode.
`*.PEN_HOLD(arg1);`	Hold a pen with `arg1`.
`*.PEN_RELEASE(arg1);`	Release the pen held by `arg1`.
`*.CS();`	Clear the trail drawn by the pen.

Section 6.2: Drawing Shapes

Next, move the human pictogram to draw a shape.

Code Example 6-2-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");

  for (let i = 0; i < 4; i++) {
    pic1.RW("LUA", 90, 1);
  }
</script>
```



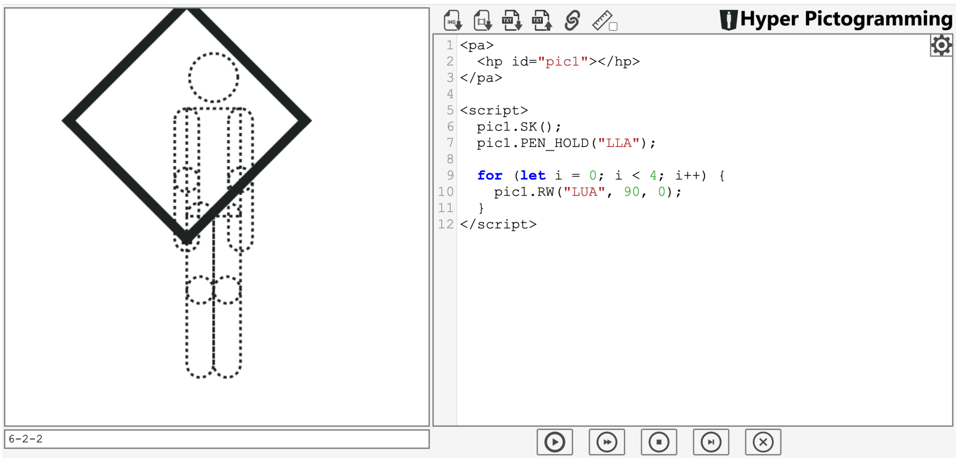
```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");

  for (let i = 0; i < 4; i++) {
    pic1.RW("LUA", 90, 1);
  }
</script>
```

By combining it with the `RW` method, you were able to draw a circle.

Next, try setting the `RW` method's `arg3` to `0` instead of `1`.

Code Example 6-2-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");

  for (let i = 0; i < 4; i++) {
    pic1.RW("LUA", 90, 0);
  }
</script>
```



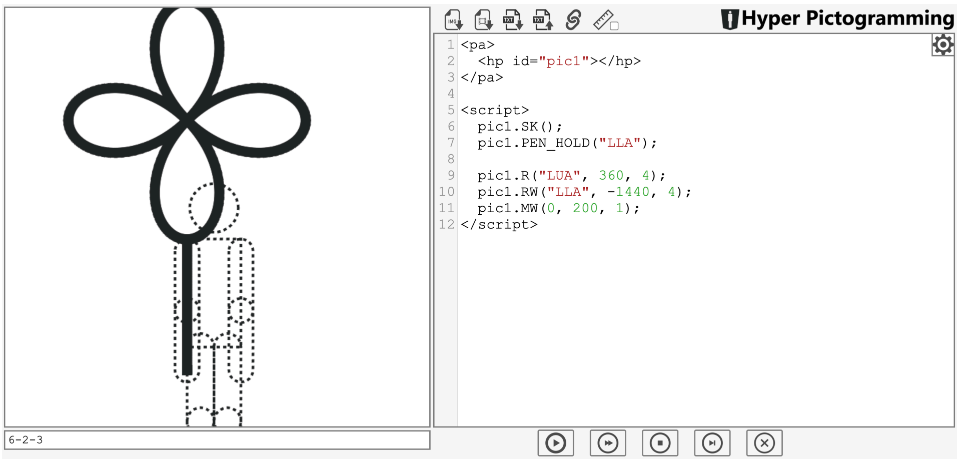
```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");

  for (let i = 0; i < 4; i++) {
    pic1.RW("LUA", 90, 0);
  }
</script>
```

You should see that a square is drawn instead of a circle. This is because `arg3=0` means instant movement (teleport). When you move instantly, a straight line is drawn between the start and end points. Repeating that creates a polygon.

By using methods such as `R`, `RW`, `M`, and `MW`, you can draw many kinds of shapes.

Code Example 6-2-3

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");

  pic1.R("LUA", 360, 4);
  pic1.RW("LLA", -1440, 4);
  pic1.MW(0, 200, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");

  pic1.R("LUA", 360, 4);
  pic1.RW("LLA", -1440, 4);
  pic1.MW(0, 200, 1);
</script>
```

You were able to draw a four-leaf clover.

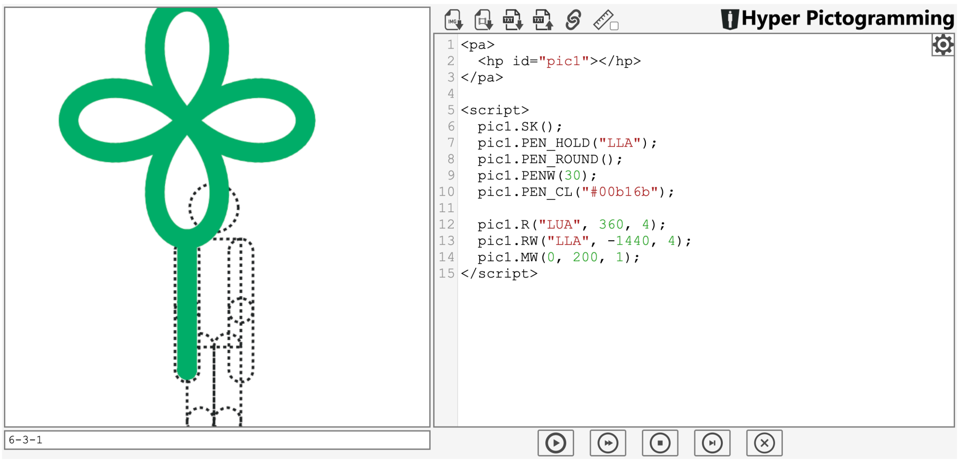
Section 6.3: Customizing the Pen

Next is how to customize the pen. Use `PEN_SQUARE`, `PEN_ROUND`, and `PEN_BUTT` to change the line cap style; `PENW` to change the pen width; and `PEN_CL` to change the pen color.

Let's customize the four-leaf clover example. Make the line caps semicircular, set the pen width to 30, and set the color to green.

Code Example 6-3-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");
  pic1.PEN_ROUND();
  pic1.PENW(30);
  pic1.PEN_CL("#00b16b");

  pic1.R("LUA", 360, 4);
  pic1.RW("LLA", -1440, 4);
  pic1.MW(0, 200, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.SK();
  pic1.PEN_HOLD("LLA");
  pic1.PEN_ROUND();
  pic1.PENW(30);
  pic1.PEN_CL("#00b16b");

  pic1.R("LUA", 360, 4);
  pic1.RW("LLA", -1440, 4);
  pic1.MW(0, 200, 1);
</script>
```

You were able to make it look more like a four-leaf clover.

Method signature	Behavior
`*.PEN_SQUARE();`	From now on, make the line caps square.
`*.PEN_ROUND();`	From now on, make the line caps round (semicircle).
`*.PEN_BUTT();`	From now on, make the line caps butt (no cap).
`*.PENW(arg1);`	Set the pen width to `arg1`. Default is `15`.
`*.PEN_CL(arg1);`	Set the pen color to `arg1`. Default depends on the pictogram area's `type`.

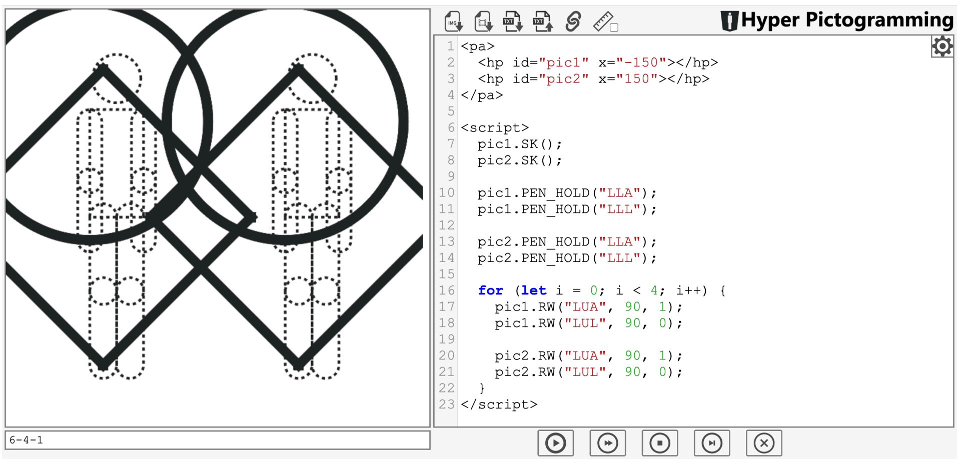
Section 6.4: Drawing Shapes with Two or More People

So far, you have drawn shapes with one human pictogram, but you can also have two or more people hold pens and draw shapes. You can also hold pens with parts other than the left hand and draw multiple shapes at the same time.

Make `pic1` and `pic2` draw a circle with the left hand and a square with the left foot, respectively.

Code Example 6-4-1

```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  pic1.SK();
  pic2.SK();

  pic1.PEN_HOLD("LLA");
  pic1.PEN_HOLD("LLL");

  pic2.PEN_HOLD("LLA");
  pic2.PEN_HOLD("LLL");

  for (let i = 0; i < 4; i++) {
    pic1.RW("LUA", 90, 1);
    pic1.RW("LUL", 90, 0);

    pic2.RW("LUA", 90, 1);
    pic2.RW("LUL", 90, 0);
  }
</script>
```

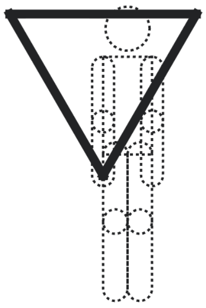


```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" x="150"></hp>
</pa>

<script>
  pic1.SK();
  pic2.SK();

  pic1.PEN_HOLD("LLA");
  pic1.PEN_HOLD("LLL");

  pic2.PEN_HOLD("LLA");
  pic2.PEN_HOLD("LLL");

  for (let i = 0; i < 4; i++) {
    pic1.RW("LUA", 90, 1);
    pic1.RW("LUL", 90, 0);

    pic2.RW("LUA", 90, 1);
    pic2.RW("LUL", 90, 0);
  }
</script>
```

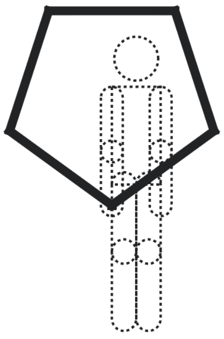
You were able to draw a circle and a square with two people.

Section 6.5: Exercises

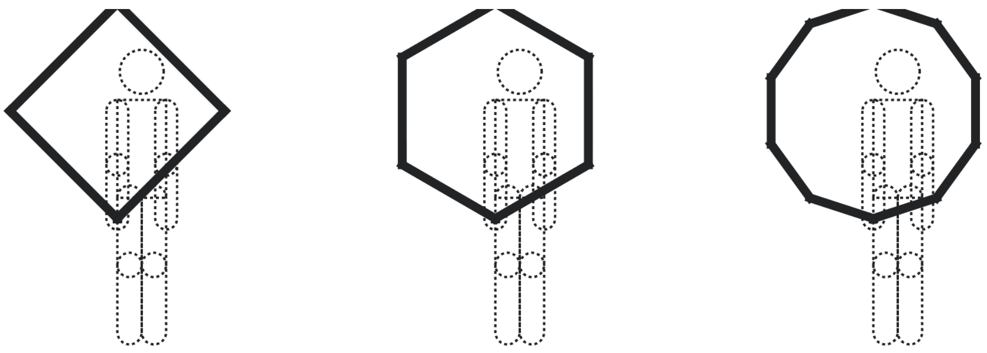
(1) Try drawing an equilateral triangle.



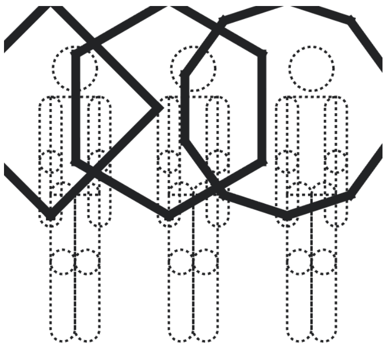
(2) Try drawing a regular pentagon.



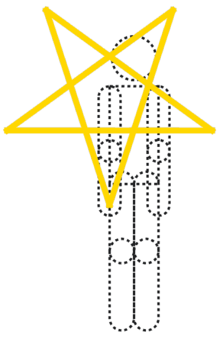
(3) Define and use a procedure that draws a regular n-gon depending on the value of argument n.



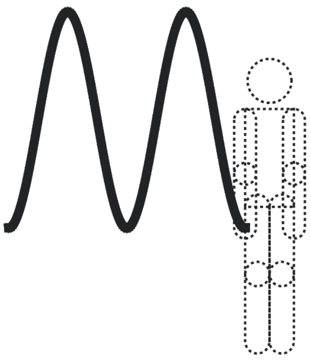
(4) Improve the code in (3) so multiple people can draw a regular n-gon at the same time.



(5) Try drawing a star.



(6) Try drawing a sine curve.



(7) Draw any shapes freely with the pen.

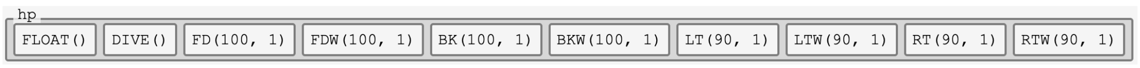
Chapter 7: Pictoswimming

This chapter covers Pictoswimming. As the name suggests, Pictoswimming is a feature that lets human pictograms swim.

Please access the URL below.

```
https://pictogramming.org/apps/hyperpictogramming/?pictoswimming=true
```

The screen is almost the same as before, but the `hp` buttons in the Code Input Assist Button Area change. As shown in the image below, buttons corresponding to the methods in this chapter are displayed.



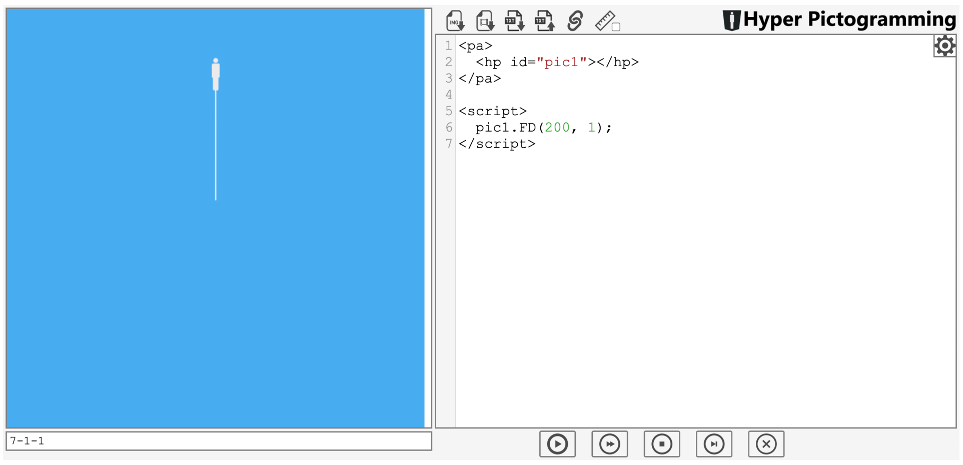
Note: You can use this chapter's methods on the previous URL as well, and you can also use the methods from the previous URL on this one.

Section 7.1: Swim

Try swimming forward. To swim forward, use the `FD` or `FDW` method (`FD` is short for Forward). In both methods, `arg1` is the distance to swim and `arg2` is the time (seconds).

Code Example 7-1-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.FD(200, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.FD(200, 1);
</script>
```

The human pictogram swam forward by 100 over 1 second.

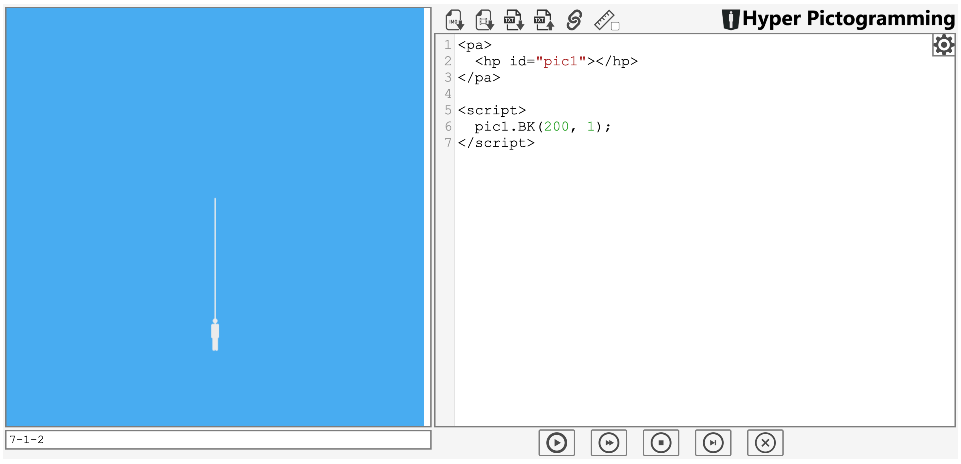
Similar to the relationship between `R` and `RW`, `FD` runs in parallel with the next command, while `FDW` blocks until the swim completes.

Method signature	Behavior
`*.FD(arg1, arg2);`	Move the human pictogram forward (in its facing direction) by distance `arg1` at constant speed over `arg2` seconds. If `arg2` is omitted, it is treated as `0`.
`*.FDW(arg1, arg2);`	Move the human pictogram forward by distance `arg1` at constant speed over `arg2` seconds. The next command will not run until the move completes. If `arg2` is omitted, it is treated as `0`.

There are also `BK` and `BKW` methods for swimming backward (`BK` is short for Backward).

Code Example 7-1-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.BK(200, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.BK(200, 1);
</script>
```

You were able to swim backward.

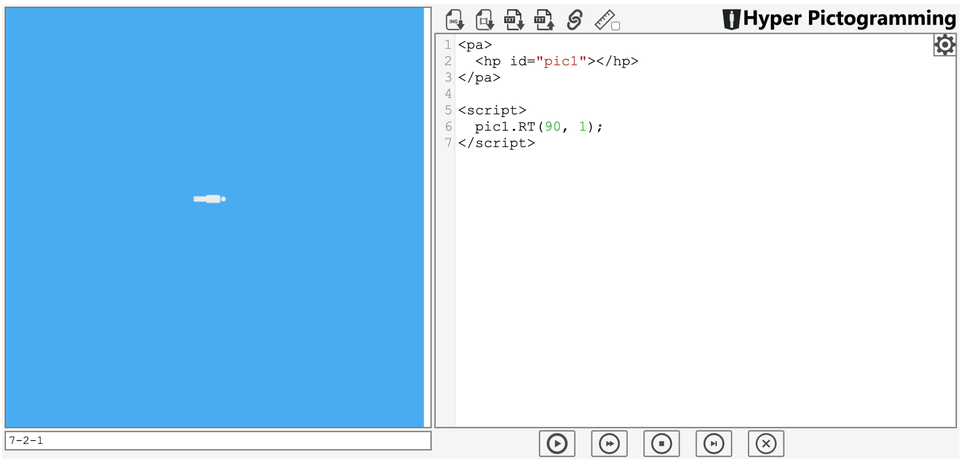
Method signature	Behavior
`*.BK(arg1, arg2);`	Move the human pictogram backward (opposite its facing direction) by distance `arg1` at constant speed over `arg2` seconds. If `arg2` is omitted, it is treated as `0`.
`*.BKW(arg1, arg2);`	Move the human pictogram backward by distance `arg1` at constant speed over `arg2` seconds. The next command will not run until the move completes. If `arg2` is omitted, it is treated as `0`.

Section 7.2: Turn

Next, rotate the human pictogram. To turn left, use `LT` or `LTW`; to turn right, use `RT` or `RTW` (`LT` is short for Left Turn and `RT` is short for Right Turn).

Code Example 7-2-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.RT(90, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.RT(90, 1);
</script>
```

You were able to rotate 90 degrees to the right in 1 second.

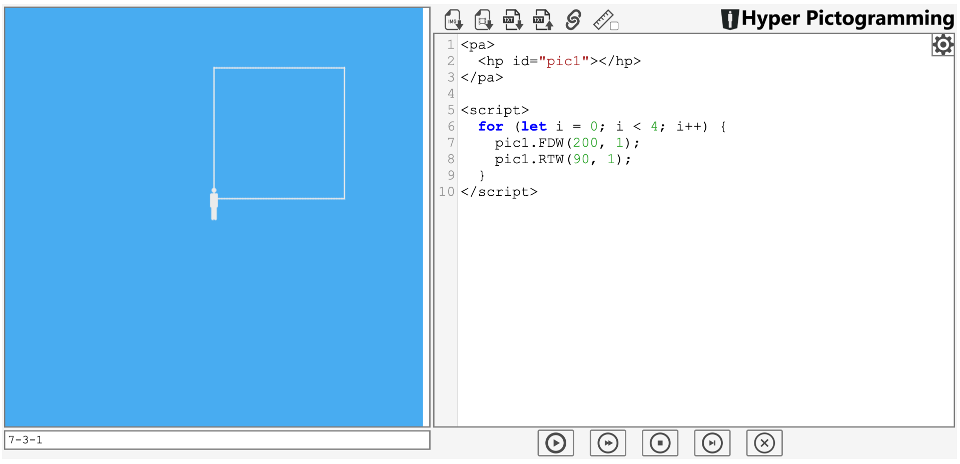
Method signature	Behavior
`*.RT(arg1, arg2);`	Rotate the human pictogram clockwise by angle `arg1` at constant angular speed over `arg2` seconds. If `arg2` is omitted, it is treated as `0`.
`*.RTW(arg1, arg2);`	Rotate the human pictogram clockwise by angle `arg1` at constant angular speed over `arg2` seconds. The next command will not run until the rotation completes. If `arg2` is omitted, it is treated as `0`.
`*.LT(arg1, arg2);`	Rotate the human pictogram counterclockwise by angle `arg1` at constant angular speed over `arg2` seconds. If `arg2` is omitted, it is treated as `0`.
`*.LTW(arg1, arg2);`	Rotate the human pictogram counterclockwise by angle `arg1` at constant angular speed over `arg2` seconds. The next command will not run until the rotation completes. If `arg2` is omitted, it is treated as `0`.

Section 7.3: Combining Swimming and Turning

Next, combine the swimming and turning methods you have used so far.

Code Example 7-3-1

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  for (let i = 0; i < 4; i++) {
    pic1.FDW(200, 1);
    pic1.RTW(90, 1);
  }
</script>
```



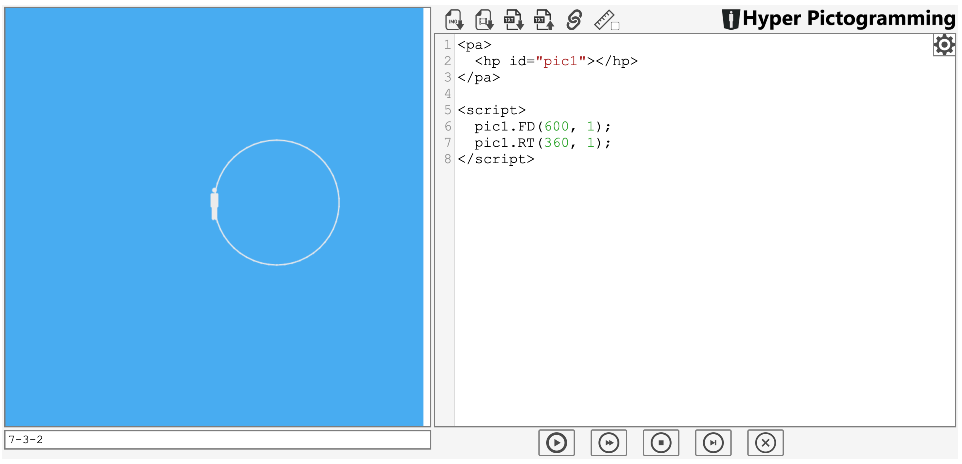
```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  for (let i = 0; i < 4; i++) {
    pic1.FDW(200, 1);
    pic1.RTW(90, 1);
  }
</script>
```

By using FDW and RTW, you were able to draw a square path.

Let’s look at one more example.

Code Example 7-3-2

```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.FD(600, 1);
  pic1.RT(360, 1);
</script>
```



```
<pa>
  <hp id="pic1"></hp>
</pa>

<script>
  pic1.FD(600, 1);
  pic1.RT(360, 1);
</script>
```

By using FD and RT, you were able to draw a circular path. Because it rotates while swimming, the trajectory becomes a circle.

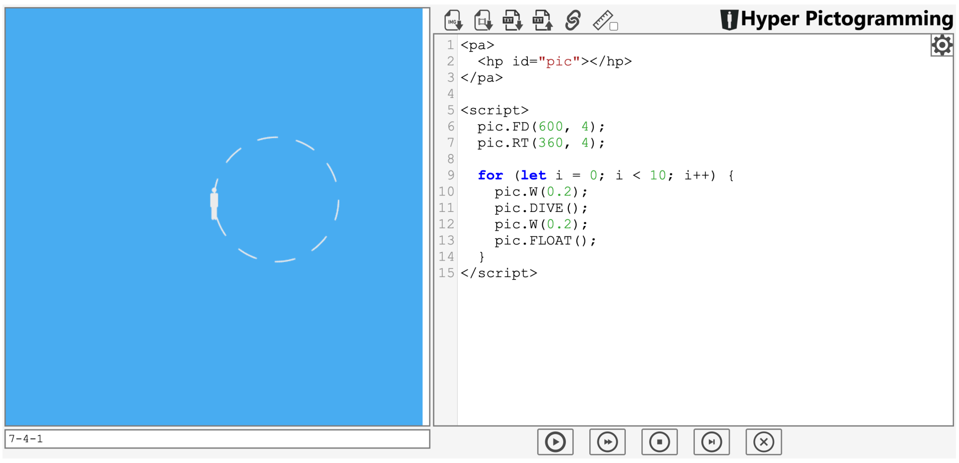
Section 7.4: Float and Dive

There are also a `FLOAT` method to make the human pictogram float up to the surface and a `DIVE` method to make it dive underwater.

Let's combine `FLOAT` and `DIVE` to draw a dotted circle.

Code Example 7-4-1

```
<pa>
  <hp id="pic"></hp>
</pa>

<script>
  pic.FD(600, 4);
  pic.RT(360, 4);

  for (let i = 0; i < 10; i++) {
    pic.W(0.2);
    pic.DIVE();
    pic.W(0.2);
    pic.FLOAT();
  }
</script>
```



```
<pa>
  <hp id="pic"></hp>
</pa>

<script>
  pic.FD(600, 4);
  pic.RT(360, 4);

  for (let i = 0; i < 10; i++) {
    pic.W(0.2);
    pic.DIVE();
    pic.W(0.2);
    pic.FLOAT();
  }
</script>
```

By repeating diving and surfacing, you were able to draw a dotted circle.

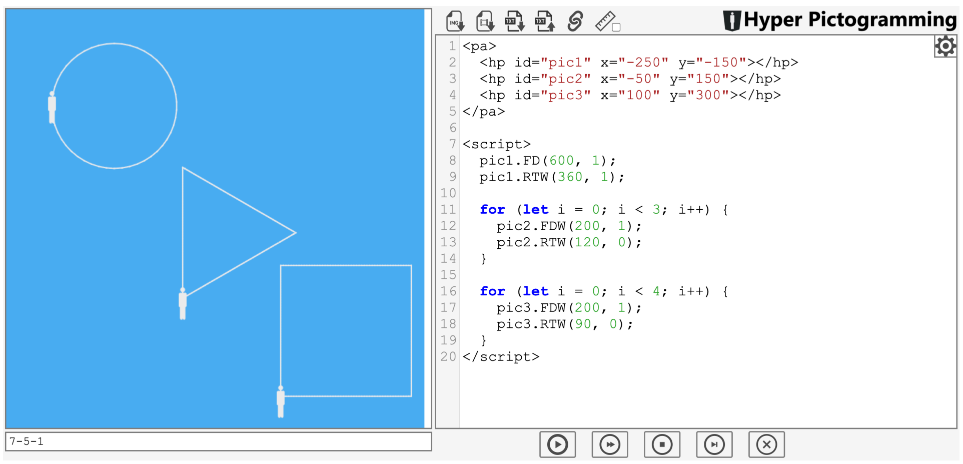
Method signature	Behavior
`*.FLOAT();`	Make the human pictogram float up to the surface.
`*.DIVE();`	Make the human pictogram dive underwater.

Section 7.5: Swim Together

As with Pictographics, Pictoswimming can also move multiple people at the same time.

Code Example 7-5-1

```
<pa>
  <hp id="pic1" x="-250" y="-150"></hp>
  <hp id="pic2" x="-50" y="150"></hp>
  <hp id="pic3" x="100" y="300"></hp>
</pa>

<script>
  pic1.FD(600, 1);
  pic1.RTW(360, 1);

  for (let i = 0; i < 3; i++) {
    pic2.FDW(200, 1);
    pic2.RTW(120, 0);
  }

  for (let i = 0; i < 4; i++) {
    pic3.FDW(200, 1);
    pic3.RTW(90, 0);
  }
</script>
```

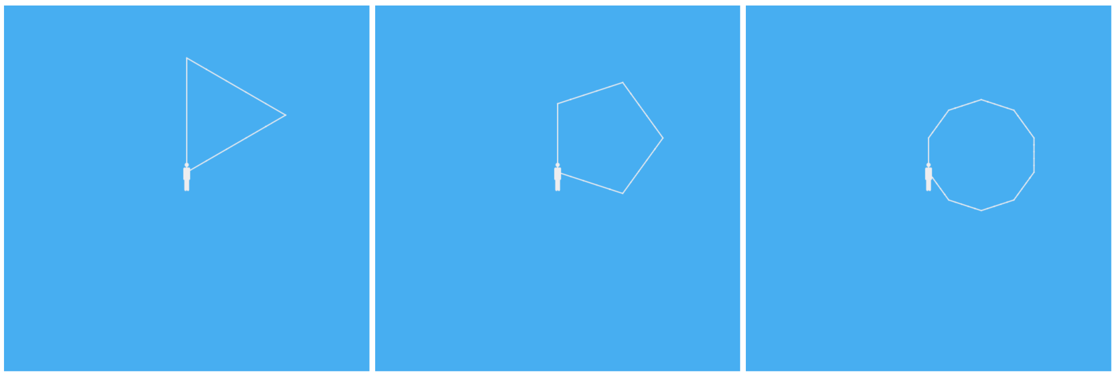


```
<pa>
  <hp id="pic1" x="-250" y="-150"></hp>
  <hp id="pic2" x="-50" y="150"></hp>
  <hp id="pic3" x="100" y="300"></hp>
</pa>

<script>
  pic1.FD(600, 1);
  pic1.RTW(360, 1);

  for (let i = 0; i < 3; i++) {
    pic2.FDW(200, 1);
    pic2.RTW(120, 0);
  }

  for (let i = 0; i < 4; i++) {
    pic3.FDW(200, 1);
    pic3.RTW(90, 0);
  }
</script>
```

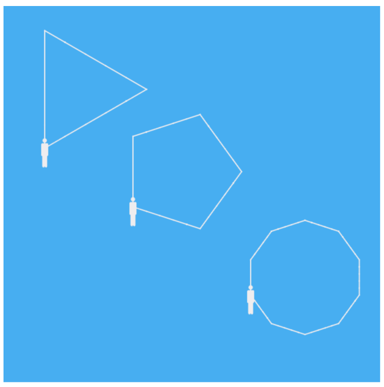
You were able to have three people draw a circle, a triangle, and a square, respectively.

Section 7.6: Exercises

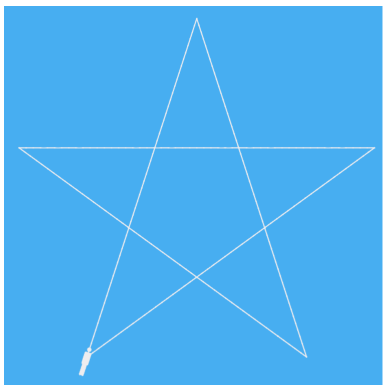
(1) Define and use a procedure that draws a regular n-gon using a swimming trajectory. Make the side length change depending on n.



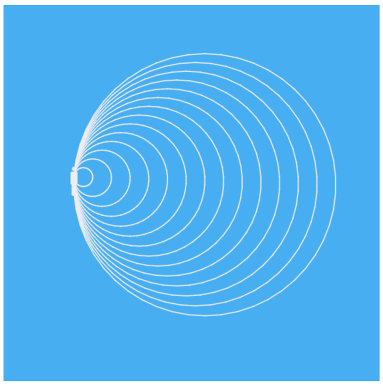
(2) Improve the code in (1) so multiple people can draw a regular n-gon at the same time.



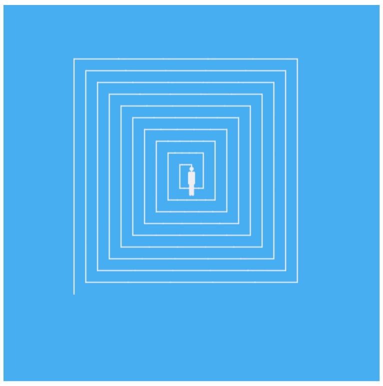
(3) Try drawing a star.



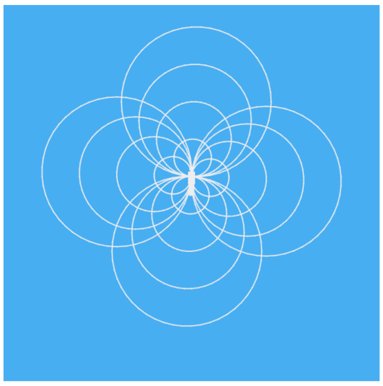
(4) Gradually make the circle smaller (or larger).



(5) Try drawing a square spiral pattern.



(6) Try drawing a flower with four people.



(7) Swim freely.

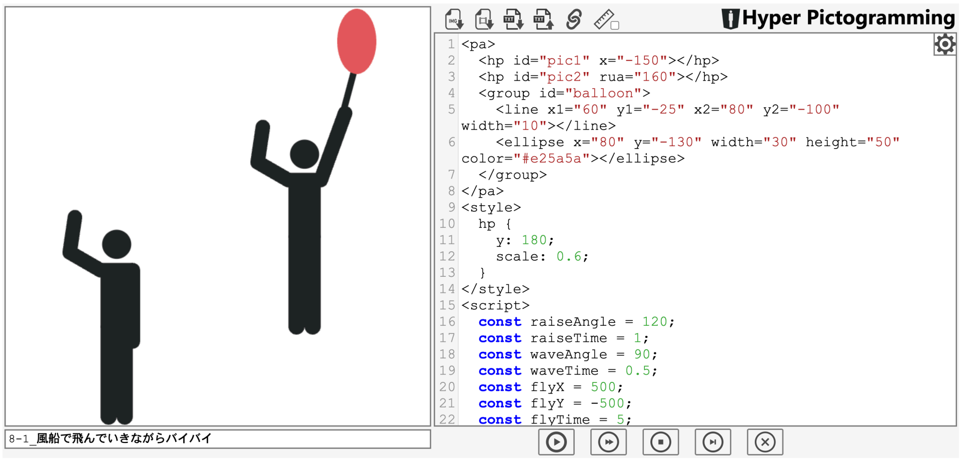
Chapter 8: Conclusion

So far, you have learned the basics of HPML, CSS, and JavaScript while creating pictograms. Next, try creating an original pictogram like the one shown below.

Code Example 8-1

```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" rua="160"></hp>
  <group id="balloon">
    <line x1="60" y1="-25" x2="80" y2="-100"
width="10"></line>
    <ellipse x="80" y="-130" width="30" height="50"
color="#e25a5a"></ellipse>
  </group>
</pa>
<style>
  hp {
    y: 180;
    scale: 0.6;
  }
</style>
<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;
  const flyX = 400;
  const flyY = -400;
  const flyTime = 5;

  pic1.byebye = byebye;
  pic1.byebye();
  pic2.fly = fly;
  pic2.fly();
  pic2.byebye = byebye;
  pic2.byebye();
  balloon.fly = fly;
  balloon.fly();

  function byebye() {
    this.RW("LUA", -raiseAngle, raiseTime);

    for (let i = 0; i < 5; i++) {
      this.RW("LLA", -waveAngle, waveTime);
      this.RW("LLA", waveAngle, waveTime);
    }

    this.RW("LUA", raiseAngle, raiseTime);
  }

  function fly() {
    this.M(flyX, flyY, flyTime);
  }
```

```
</script>
```



```
<pa>
  <hp id="pic1" x="-150"></hp>
  <hp id="pic2" rua="160"></hp>
  <group id="balloon">
    <line x1="60" y1="-25" x2="80" y2="-100" width="10"></line>
    <ellipse x="80" y="-130" width="30" height="50" color="#e25a5a"></ellipse>
  </group>
</pa>
<style>
  hp {
    y: 180;
    scale: 0.6;
  }
</style>
<script>
  const raiseAngle = 120;
  const raiseTime = 1;
  const waveAngle = 90;
  const waveTime = 0.5;
  const flyX = 500;
  const flyY = -500;
  const flyTime = 5;
```

This is an animated pictogram where, out of two human pictograms, one flies away with a balloon while the other stays and waves goodbye.

Is there anything where a pictogram could help communicate information well? Using what you have learned so far, try creating your own original pictogram.