pdl primitives

Version 1.07

Ian Lawrence Friday, 15 November 2019

Diagrams by code

PID

Physics interactive diagrams: the what and why

These are web-friendly physics diagrams which are built in an environment that draws on p5js.

The diagrams are constructed from a set of primitives, many being physics representations such as acceleration arrows, together with a standard set of controls, such as sliders, buttons and pucks (2D sliders). In addition html text entities are co-opted to allow both CSS control of text and to co-opt the physics markup language in rendering technical physics text. These are glued together with the scripting language in p5js, which means that the diagrams can be interactive.

This use of a framework (physics diagram language, pdl) leads to an inheritance hierarchy across a suite of diagrams, which makes consistency and editability both possible and reliable.

The diagram file

```
function preload() {
    // fonts, SVG graphics images pre-loaded here
}
function setup(){
    // everthing that needs doing once, on first loading
}
function draw() {
    // everything that
    }
```

Invoking the diagram

Each diagram lives in its own folder. The folder has a meaningful name, which is designed to be sit as the terminus of a pathname, so does not contain spaces or other punctuation. Inside the folder is an index.html file, which loads the environment and the diagram file.

```
<!DOCTYPE html>
<html>
  <head>
    <title>start</title>
     <link rel="stylesheet" type="text/css" href="../__PDLenviron-
ment/PDL.css">
    <script src="../__PDLenvironment/p5.min.js"></script>
    <script src="../__PDLenvironment/p5.dom.min.js"></script>
    <script src="../__PDLenvironment/p5.sound.min.js"></script>
        <script  src="../__PDLenvironment/PDLglobalvariables.js"></
script>
     <script src="../__PDLenvironment/PDLgraphicannotations.js"></
```

```
script>
        <script  src="../__PDLenvironment/PDLwordannotations.js"></
script>
    <script src="../__PDLenvironment/PDLphysics.js"></script>
    <script src="../__PDLenvironment/xregexp-all.js"></script>
     <script src="../__PDLenvironment/slider.js"></script>
    <script src="../__PDLenvironment/pmlparser.js"></script>
    <script src="diagram.js"></script>
  </head>
  <body>
  </body>
</html>
```

As a consequence entering the URL of the folder into a browser window executes the sketch.

Incorporating PID into a page

In the physics markup language (pml) these are inserted through

```
InsertGraphic{.*PID}{NNN}{NNN}
```

for physics interactiev diagrams, or

```
InsertGraphic{.*DIP}{NNN}{NNN}
```

for discussing instances in physics.

PSS

Physics Story spaces: The what and the why

The physics story space lays out an alternating series of panes and transitions from top left to bottom right to give a graphic novel-like experience to the narrative. The resources drawn on by the system are a structured text file, a series of prepared graphics, and a JavaScript/HTML/CSS framework. The core of the system is a parser that takes a structured text file and lays out the graphics as alternating panes and transitions in a rendering framework.

The text file

`Filename: XXX`

`WebTitle: XXX`

`TransitionSize{NN}`

`StepPane{thename}{URL}{NNN}{NNN}`

`TransitionStep{URL}{transitionoffset}`

`StepPane{anothername}{URL}{NNN}{NNN}`

`TransitionStep{URL}{transitionoffset}`

Elements of the text file

This is written in a simplified markup - set of document data

`Filename: XXX`

`WebTitle: XXX`

`TransitionSize{NN}`

and a sequence of content elements constructed from:

`StepPane{name}{URL}{width}{height}`

where

- name sets the id of the StepPane (no spaces, lower case, unique within this story space)
- URL sets the content of the pane
- width is an integer that sets the width of the pane in pixel
- height is an integer that sets the height of the pane in pixel

and

`TransitionStep{URL}{transitionoffset}`

where

- URL sets the content of the transition
- transitionoffset is a string which determines the location of the following StepPane, and therefore also the location of this TransitionStep. The string has one of the following values: DownLeft, DownRight, DownCentre, RightTop, RightBottom, RightCentre. In each case the first word in the CamelCase pair indicates the displacement (Down or Right) and the second word indicates how the StepPane is to

aligned to the previous pane(align tops, centres, left hand edges, right hand edges, bottoms).

Each pair of StepPanes are separated by a TransitionStep.

Incorporating into a page

In the physics markup language (pml) these are inserted through

`InsertGraphic{NamePSS}`

Invoking directly

`http://supportingphysicsteaching.net/ShowStory.html?file=http://supportingphysicsteaching.net/FoPSSMakingHolesMel.md`

So the text file is passed to the framework as a URL file parameter. Often there is also a redirect, such as `http://supportingphysicsteaching.net/FoPSSMakingHolesMel`

This is currently a bit clumsy for development / historical reasons, and might, at some stage get rewritten to be closer to the PID model.

Notes

- Non-linear routing was considered, but splitting and joining narrative routes would require the panes to be labelled, so reducing the simplicity
- Thoughts could be switched to a different track by hard-wiring links into the graphics
- The simple flow assist the reader — and we should not overload the reader unnecessarily
- The "waste" of screen space is not really an issue as it might be on paper, so we do not need fly back to the left-hand margin in order to "use" the rest of the page
- Moving left-right and top-bottom are simple sequences, well established in western culture as a narrative guide
- Hat tip to Scott McCloud's infinite canvas
- This is currently built on impressjs as a framework, but only uses a small fraction of the capabilities.
- Hovercraft, as a restructured text authoring tool for impressjs(corroborated my view that this kind of thing should be possible).
- The help text should be easy to find in the html, so as to be easily adaptable
- The essence is that a simple-to-author and adapt text file should be parsed to navigate alternating panes and transitions.

PRS

Physics roaming spaces: The what and the why

The physics roaming space lays out an array of panes (and optional transitions to give an explorable paned diagram. The resources drawn on by the system are a structured text file, a series of prepared graphics, and a JavaScript/HTML/CSS framework. The core of the system is a parser that takes a structured text file and lays out the panes and transitions in a rendering framework.

The text file

```
FileName: filename.md
WebTitle: an explanatory phrase
TransitionSize{50}
ExplorePane{PaneId}{WWW,HHH}{URL}
```

Elements of the text file

This is written in a simplified markup - set of document data

```
Filename: XXX
WebTitle: XXX
TransitionSize{NN}
```

and a sequence of content elements constructed from:

```
ExplorePane{PaneId}{WWW,HHH}{URL}
```

where

- name sets the id of the StepPane (no spaces, lower case, unique within this story space)
- URL sets the content of the pane
- width is an integer that sets the width of the pane in pixel
- height is an integer that sets the height of the pane in pixel

and

```
TransitionStep{PaneId-XN}{PaneId-XN}{URL}
```

where

- URL sets the content of the transition
- PaneId sets the panes that are linked
- X is the pane direction, and can be one of A, B, L, R for above, below, left or right
- N is the Button offset, and can be one of 0,3,5,7,9 for the tenths of the pane along from the specified edge

Each pair of ExplorePanes are separated by a TransitionStep, which may contain a blank graphic.

Incorporating into a page

In the physics markup language (pml) these are inserted through

`InsertGraphic{NamePRS}`

Invoking directly

`http://supportingphysicsteaching.net/ShowRoam.html?file=http://supportingphysicsteaching.net/SPTGraphics/PRSDrivers/EeStoreCalculations.md`

So the text file is passed to the framework as a URL file parameter. Often there is also a redirect, such as `http://supportingphysicsteaching.net/FoPRSMakingHoles`

This is currently a bit clumsy for development / historical reasons, and might, at some stage get rewritten to be closer to the PID model.

Notes

- Designed to allow roaming over a landscape
- Hat tip to Scott McCloud's infinite canvas
- This is currently built on impressjs as a framework, but only uses a small fraction of the capabilities.
- The essence is that a simple-to-author and adapt text file should be parsed to navigate panes and transitions.

PCS

Physics comic spaces: The what and the why

The physics roaming space lays out an array of panes (and optional transitions to give an explorable paned diagram. The resources drawn on by the system are a structured text file, a series of prepared graphics, and a JavaScript/HTML/CSS framework. The core of the system is a parser that takes a structured text file and lays out the panes and transitions in a rendering framework.

The text file

```
FileName: filename.md
WebTitle: An explanatory phrase
TransitionSize{0}
ExplorePane{PaneId}{WWW,HHH}{URL}
```

Elements of the text file

This is written in a simplified markup - set of document data

```
Filename: XXX
WebTitle: XXX
TransitionSize{0}
```

and a sequence of content elements constructed from:

```
ExplorePane{PaneId}{WWW,HHH}{URL}
```

where

- name sets the id of the StepPane (no spaces, lower case, unique within this story space)
- URL sets the content of the pane
- width is an integer that sets the width of the pane in pixel
- height is an integer that sets the height of the pane in pixel

and

```
TransitionStep{PaneId-XN}{PaneId-XN}{}
```

where

- The transition contents are always null
- PaneId sets the panes that are linked
- X is the pane direction, and can be one of A, B, L, R for above, below, left or right
- N is the Button offset, and can be one of 0,3,5,7,9 for the tenths of the pane along from the specified edge

Each pair of ExplorePanes are orientated with respect to a blank TransitionStep.

Incorporating into a page

In the physics markup language (pml) these are inserted through

`InsertGraphic{NamePCS}`

Invoking directly

`http://supportingphysicsteaching.net/SPTGraphics/__PDLenvironment/ShowComic.html?file=http://supportingphysicsteaching.net/SPTGraphics/PCSDrivers/EeLightingX.md`

So the text file is passed to the framework as a URL file parameter. Often there is also a redirect, such as `http://supportingphysicsteaching.net/FoPRSMakingHoles`

This is currently a bit clumsy for development / historical reasons, and might, at some stage get rewritten to be closer to the PID model.

Notes

- Designed to allow roaming over comic panes
- Hat tip to Scott McCloud's infinite canvas
- This is currently built on impressjs as a framework, but only uses a small fraction of the capabilities of impressjs.
- The essence is that a simple-to-author and adapt text file should be parsed to navigate panes and transitions.

PTS

Physics teachable sequences: What and why

The physics teachable space lays out a sequence of triplets of panes(teacher interventio, children reasoning, supporting notes) and standardised transitions to give an explorable view of a teachable sequence. The resources drawn on by the system are a structured text file, a series of prepared graphics, and a JavaScript/HTML/CSS framework. The core of the system is a parser that takes a structured text file and lays out the panes and transitions in a rendering framework.

The text file

`FileName: filename.md`

`WebTitle: An explanatory phrase`

`TransitionSize{50}`

`ExplorePane{PaneId}{WWW,HHH}{URL}`

Elements of the text file

This is written in a simplified markup - set of document data

`Filename: XXX`

`WebTitle: XXX`

`TransitionSize{NN}`

followed by an introduction and a transition

`ExplorePane{Introduction}{600,600}{URL}`

`TransitionStep{Introduction-A0}{TeachingInterventionStep01-B0}{../__PDLgraphics/transitionStepDown.svg}`

and then a structured sequence of content elements constructed from:

`ExplorePane{PaneId}{WWW,HHH}{URL}`

where

- name sets the id of the ExplorePane (no spaces, lower case, unique within this story space)
- URL sets the content of the pane
- width is an integer that sets the width of the pane in pixel
- height is an integer that sets the height of the pane in pixel

and

`TransitionStep{PaneId-XN}{PaneId-XN}{URL}`

where

- URL sets the content of the transition
- PaneId sets the panes that are linked
- X is the pane direction, and can be one of A, B, L, R for above, below, left or right

- N is the Button offset, and can be one of 0,3,5,7,9 for the tenths of the pane along from the specified edge

Each triplet of ExplorePanes are separated by a pair of TransitionSteps to link that triplet and a further TransitionStep to link to the next triplet. Here is an example, showing a linked pair of triplets:

```
ExplorePane{TeachingInterventionStep01}{800,600}{../../Ra02PN-nugget07.html}
ExplorePane{PupilReasoningStep01}{800,600}{../../Ra02PN-nugget02.html}
ExplorePane{SupportingRationaleStep01}{600,400}{../../Ra01PN-nugget07.html}

TransitionStep{TeachingInterventionStep01-L5}{PupilReasoning-Step01-R5}{../__PDLgraphics/transitionInterveneRight.svg}
TransitionStep{TeachingInterventionStep01-R7}{SupportingRatio-naleStep01-L3}{../__PDLgraphics/transitionStepRight.svg}

TransitionStep{PupilReasoningStep01-A9}{TeachingIntervention-Step02-B3}{../__PDLgraphics/transitionInterveneDown.svg}

ExplorePane{TeachingInterventionStep02}{800,600}{../../Ra02PN-nugget07.html}
ExplorePane{PupilReasoningStep02}{800,600}{../../Ra02PN-nugget02.html}
ExplorePane{SupportingRationaleStep02}{600,400}{../../Ra01PN-nugget07.html}

TransitionStep{TeachingInterventionStep02-L5}{PupilReasoning-Step02-R5}{../__PDLgraphics/transitionInterveneRight.svg}
TransitionStep{TeachingInterventionStep02-R7}{SupportingRatio-naleStep02-L3}{../__PDLgraphics/transitionStepRight.svg}
```

Incorporating into a page

In the physics markup language (pml) these are inserted through

```
InsertGraphic{NamePTS}
```

Invoking directly

```
http://supportingphysicsteaching.net/SPTGraphics/__PDLenvironment/ShowTeachableSequence.html?file=http://supportingphysicsteach-ing.net/SPTGraphics/PTSDrivers/ProtoTeachableSequence.md
```

So the text file is passed to the framework as a URL file parameter. Often there is also a redirect, such as `http://supportingphysicsteaching.net/FoPRSMakingHoles`

This is currently a bit clumsy for development / historical reasons, and might, at some stage get rewritten to be closer to the PID model.

Notes

- Designed to allow roaming over a landscape
- Hat tip to Scott McCloud's infinite canvas
- This is currently built on impressjs as a framework, but only uses a small fraction of the capabilities.
- The essence is that a simple-to-author and adapt text file should be parsed to navigate panes and transitions.

physics qualitative

absorber

primitive and parameters

`absorber(quality, awidth, aheight)`

description

A rectangular block on the canvas which represents an absorber. The different qualities provide a variation in appearance to show differences in the absorption.

parameters

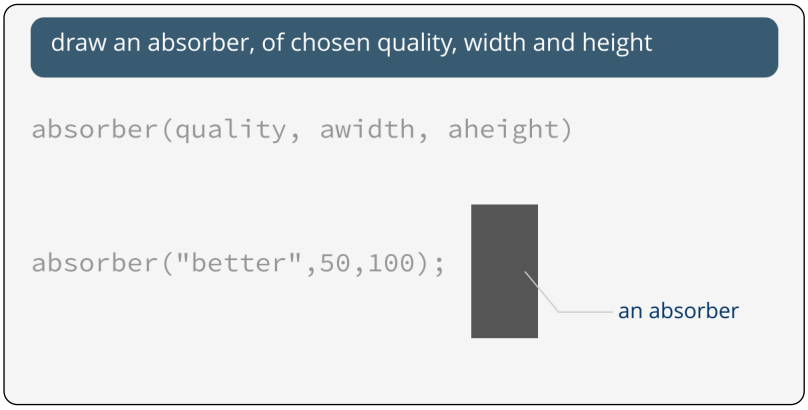
`quality`: poor; good; better; perfect.

`awidth`: width of absorber in pixel

`aheight`: width of absorber in pixel

examples

```
absorber("poor",blockwidth, blockheight);
absorber("good",10, 50);
```



bulb

primitive and parameters

`bulb(label)`

description

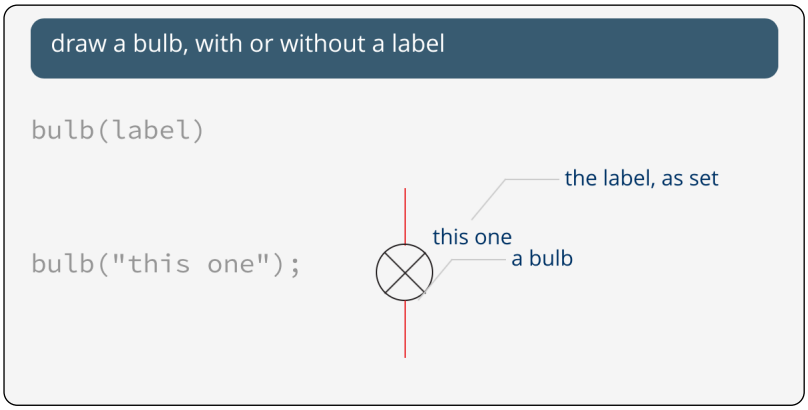
Draws a bulb with an optional label and two connecting wires.

parameters

`label`: the label to show next to the bulb. Enter "" for no label.

examples

```
bulb("b1");
bulb("");
bulb("this one and no other");
```



battery

primitive and parameters

`battery (label)`

description

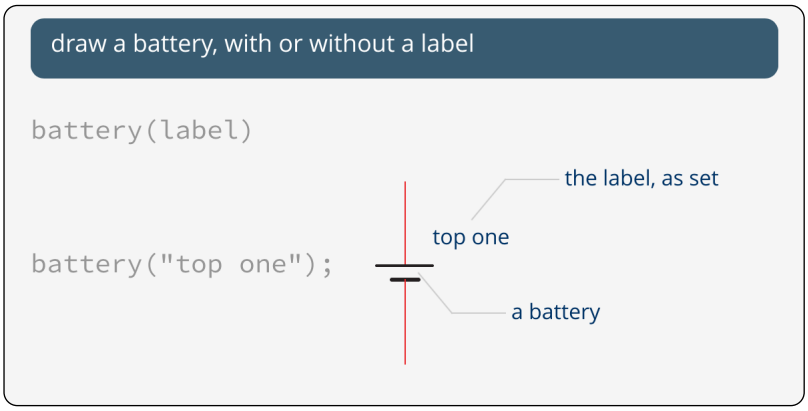
Draws a battery with an optional label and two connecting wires.

parameters

`label:` the label to show next to the battery. Enter "" for no label.

examples

```
battery("pd1");
battery("");
battery("imagined as having no internal resistance");
```



resistor

primitive and parameters

resistor `(label)`

description

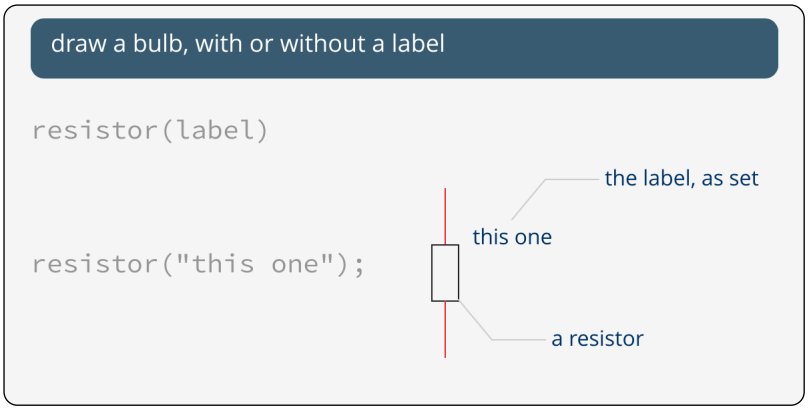
Draws a resistor with an optional label and two connecting wires.

parameters

`label`: the label to show next to the resistor. Enter "" for no label.

examples

```
resistor ("R1");
resistor ("");
resistor ("this one and not the other one");
```



circuitSimple

primitive and parameters

`circuitSimple(kind)`

description

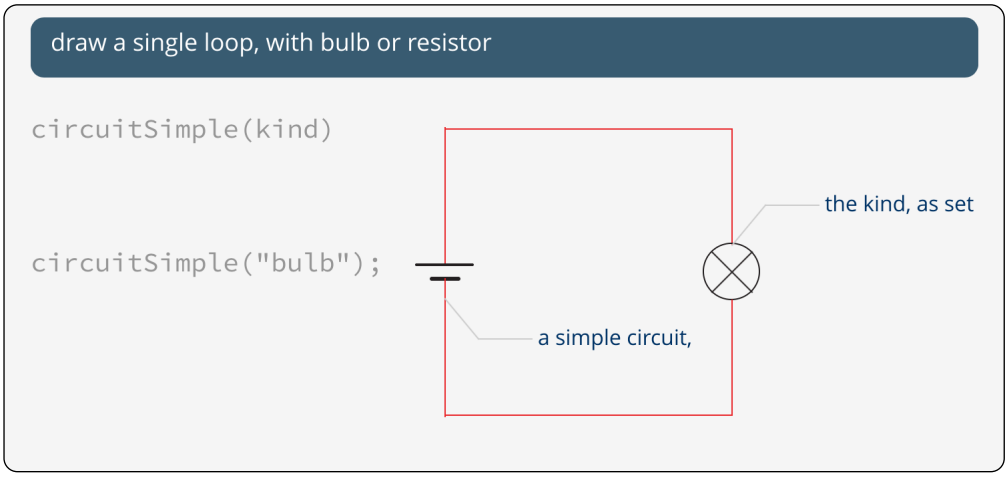
Creates a simple circuit of specified kind – one of: bulb; resistor.

parameters

`kind` : bulb; resistor

examples

```
circuitSimple("bulb");
circuitSimple("resistor");
```



circuitSimpleStretched

primitive and parameters

`circuitSimpleStretched(kind)`

description

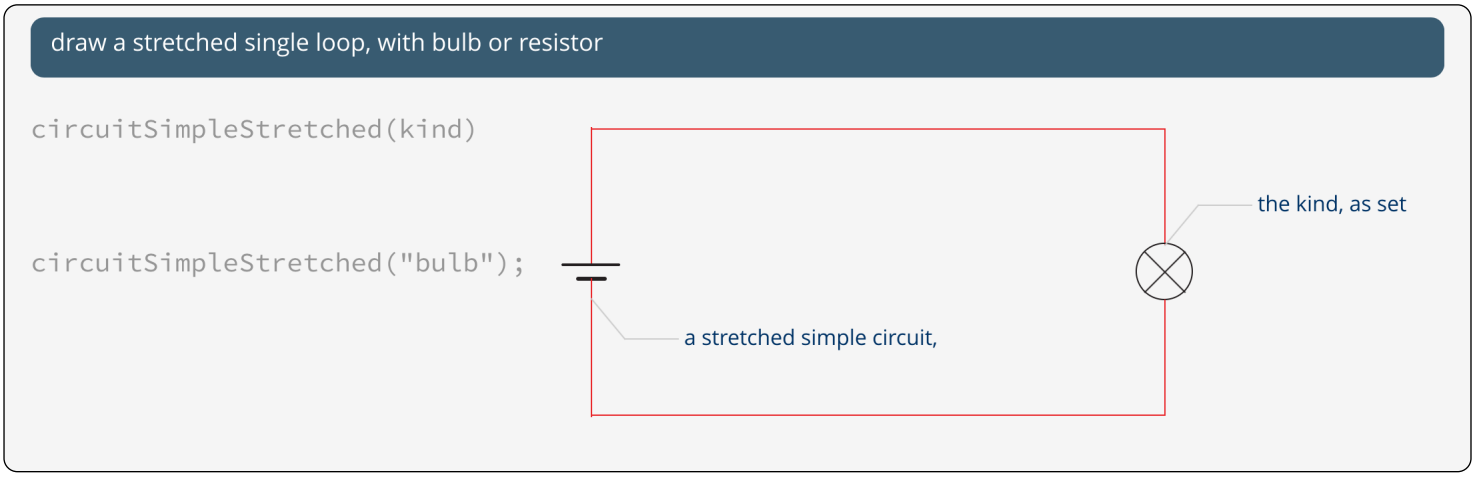
Creates a simple circuit of specified kind – one of: bulb; resistor.

parameters

`kind` : bulb; resistor

examples

```
circuitSimpleStretched("bulb");
circuitSimpleStretched("resistor");
```



circuitParallel

primitive and parameters

`circuitParallel(kind)`

description

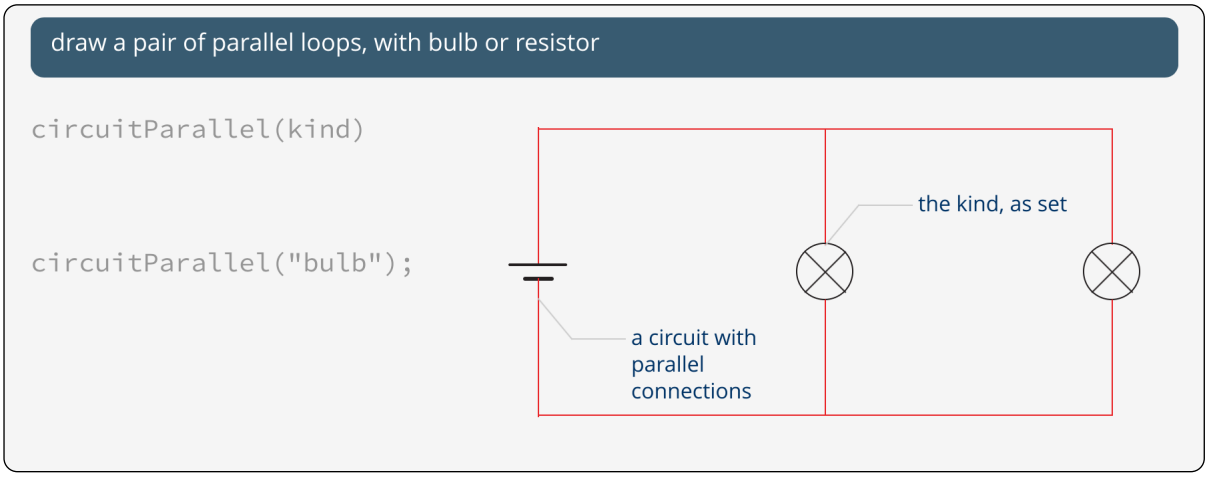
Creates a circuit with parallel connections of specified kind – one of: bulb; resistor.

parameters

`kind` : bulb; resistor

examples

```
circuitParallel("bulb");
circuitParallel("resistor");
```



circuitSeries

primitive and parameters

`circuitSeries(kind)`

description

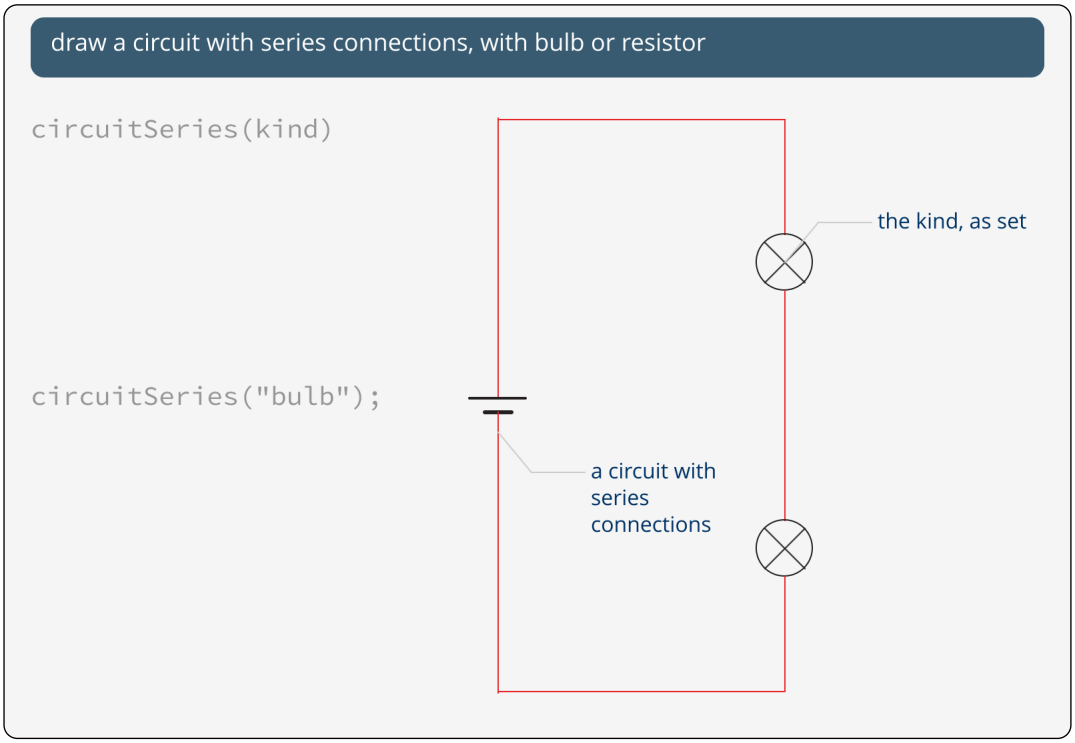
Creates a circuit with series connections of specified kind – one of: bulb; resistor.

parameters

`kind` : bulb; resistor

examples

```
circuitSeries("bulb");
circuitSeries("resistor");
```



loopOne

primitive and parameters

`loopOne()`

description

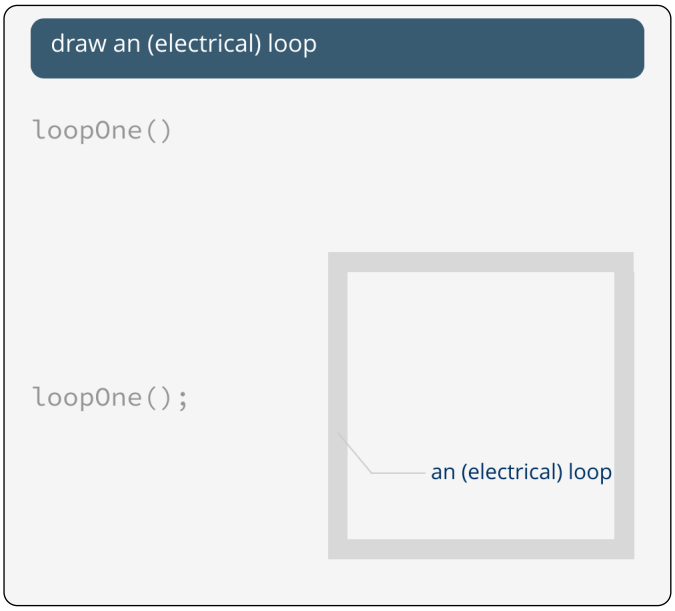
Place a loop, designed to sit behind a `circuitSimple`.

parameters

none

examples

`loopOne()`:



loopTwo

primitive and parameters

`loopTwo()`

description

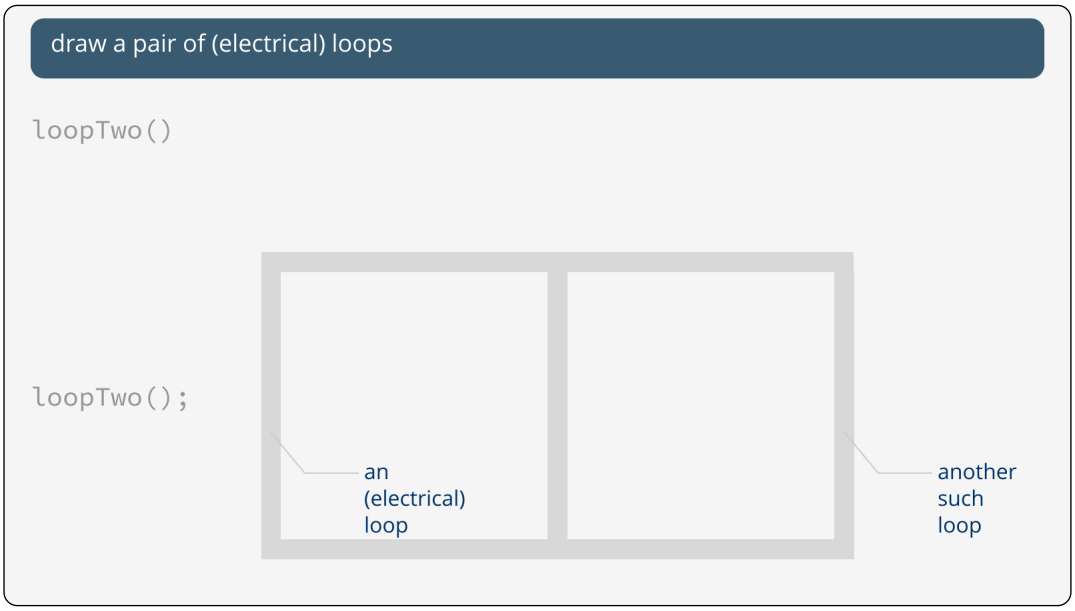
Place two loops, designed to sit behind a `circuitParallel`.

parameters

none

examples

`loopTwo()`:



loopOneStretched

primitive and parameters

`loopOneStretched()`

description

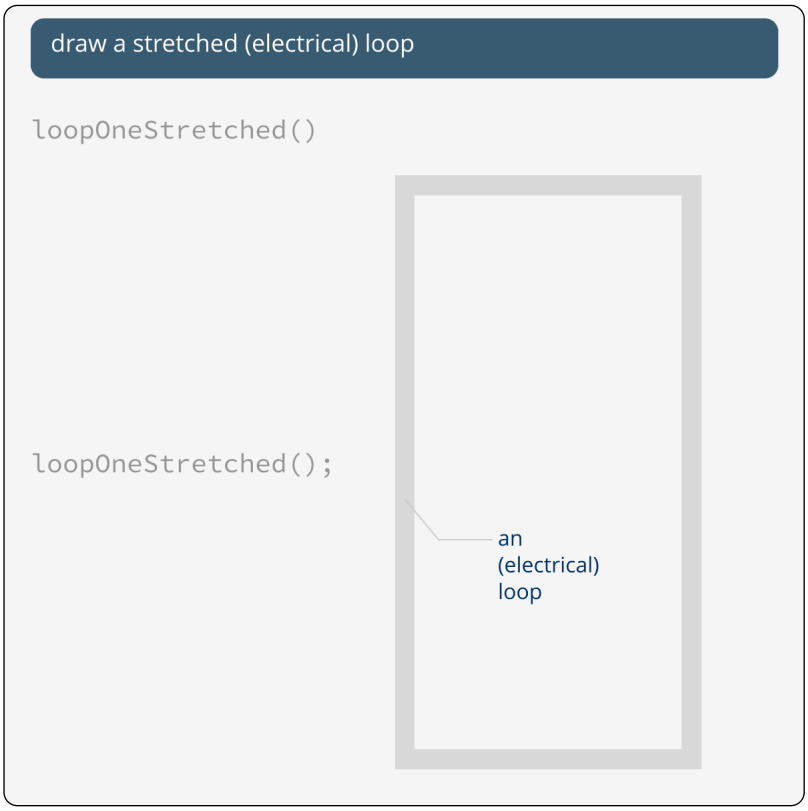
Place a stretched loop, designed to sit behind a `circuitSimpleStretched`.

parameters

none

examples

`loopOneStretched()`:



prism

primitive and parameters

`prism(pheight, pbase, angle)`

description

Draws an optical prism, which can be rotated. The origin of the prism is at the centre of the base of the prism.

parameters

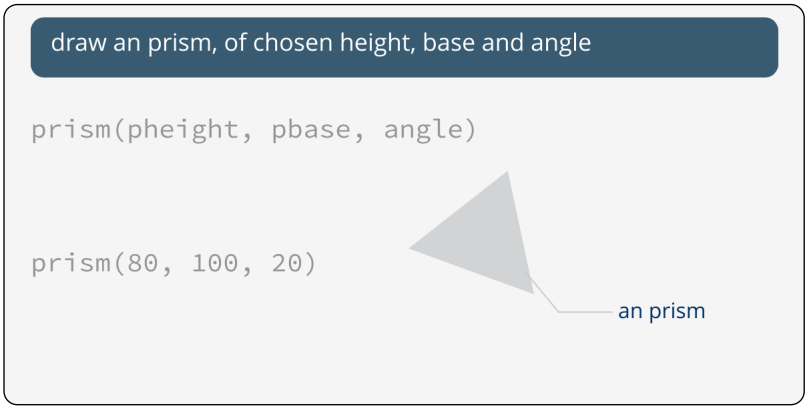
`pheight:` height of prism in pixel

`pbase:` width of prism in pixel

`angle`: rotation of base of prism in degrees

examples

```
prism(60, 40, -30);
prism(80, 100, 20);
```



reflector

primitive and parameters

`reflector(quality,angle,length)`

description

draws a reflector, which can be rotated, of chosen quality and length.

parameters

`quality:` poor; good; better; perfect

`angle:` rotation in degrees

`length:` length in pixel

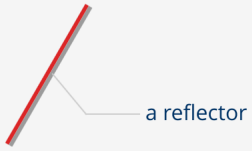
examples

```
reflector("poor", -10, 200);
reflector("good", 30, 120);
```

draw an reflector, of chosen quality,angle and length

```
reflector(quality,angle,length)


reflector("good", 30, 120);
```



convexLens

primitive and parameters

`convexLens(lheight, thickness, lines)`

description

Draws a convex lens of chosen height and thickness. You can chose whether to show the construction lines or not. The focal length is 75 pixel. The diagram is drawn from the optical centre of the lens.

parameters

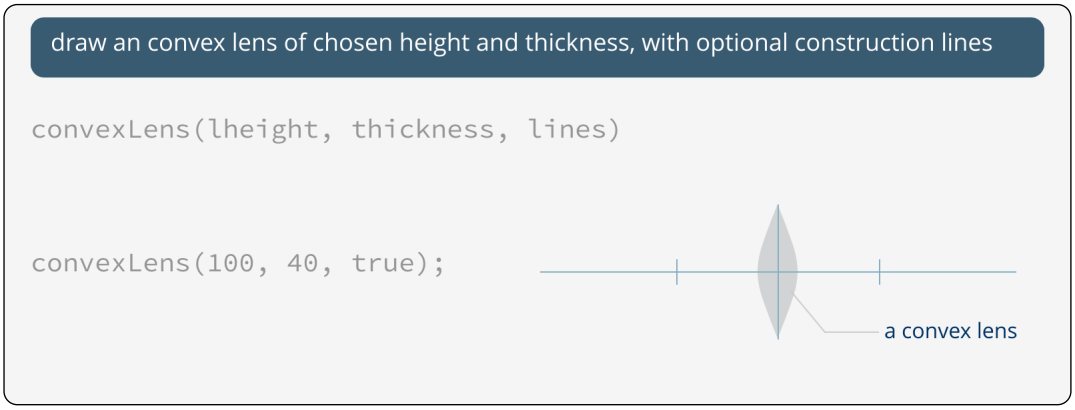
`lheight:` height of the lens in pixel

`thickness:` thickness of the lens in pixel

`lines`: true or false, to show the construction lines, or not

examples

```
convexLens(200, 20, false);
convexLens(100, 40, true);
```



concaveLens

primitive and parameters

`concaveLens(lheight, thickness, lines)`

description

Draws a concave lens of chosen height and thickness. You can chose whether to show the construction lines or not. The focal length is 75 pixel. The diagram is drawn from the optical centre of the lens.

parameters

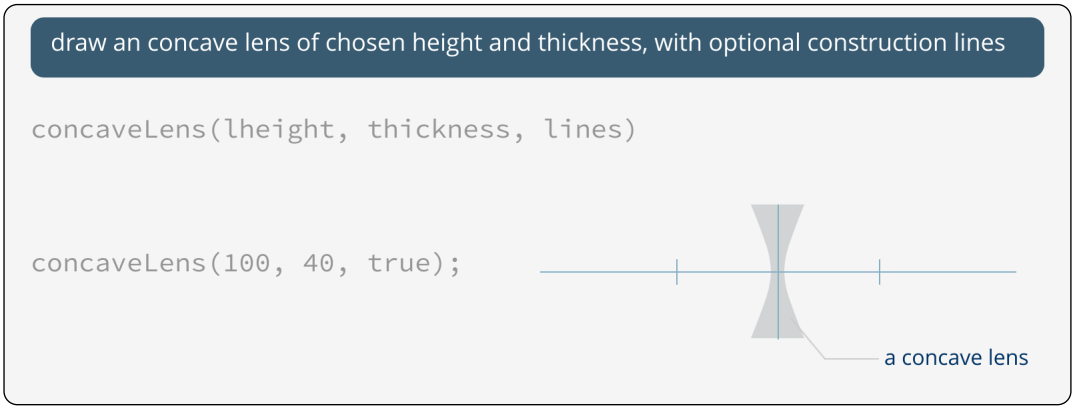
`lheight:` height of the lens in pixel

`thickness:` thickness of the lens in pixel

`lines`: true or false, to show the construction lines, or not

examples

```
concaveLens(200, 20, false);
concaveLens(100, 40, true);
```



transducer

primitive and parameters

`transducer(tcolor,angle)`

description

draw a transducer of specified colour, at an angle specified in degrees. The transducer is drawn from the centre of the emitting or receiving arc.

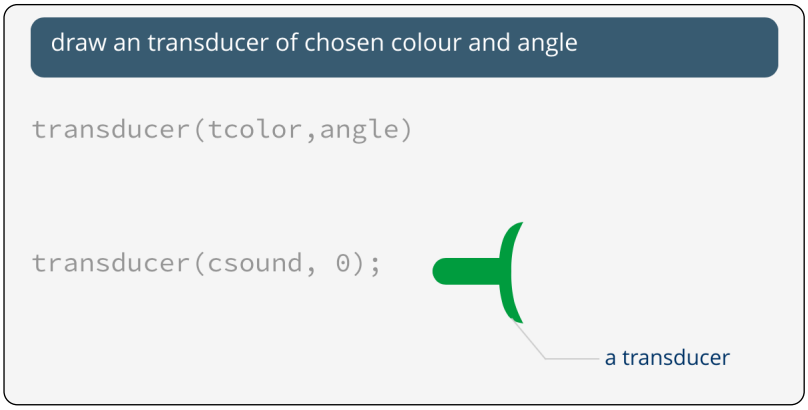
parameters

`tcolor:` colour of the transducer

`angle`: angle of the transducer in degrees

examples

```
transducer(clight, 0);
transducer(cBlack, 180);
transducer(csound, 0);
```



eye

primitive and parameters

`eye(angle)`

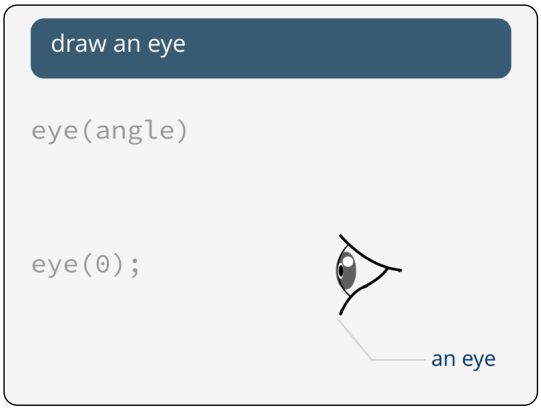
description

Draw an eye, centred on the pupil, rotated by a chosen angle.

parameters

`angle`: rotation of eye in degrees

examples



waypoint

primitive and parameters

`waypoint(wcolour)`

description

Draws a waypoint, centred on the base of the flagpole. The suite of colours designed to represent contributions might find a good use here: ccongreen,cconpink,cconorange,cconlight-green,ccongray,cconpurple

parameters

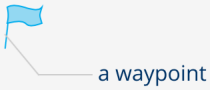
`wcolour:` any colour.

examples

draw a waypoint

```
waypoint(wcolour)


waypoint(cconcyan);
```



magnet

primitive and parameters

`magnet(angle, hidelabels)`

description

Draws a bar magnet, with or without the labels on the poles, rotated by the chosen angle.

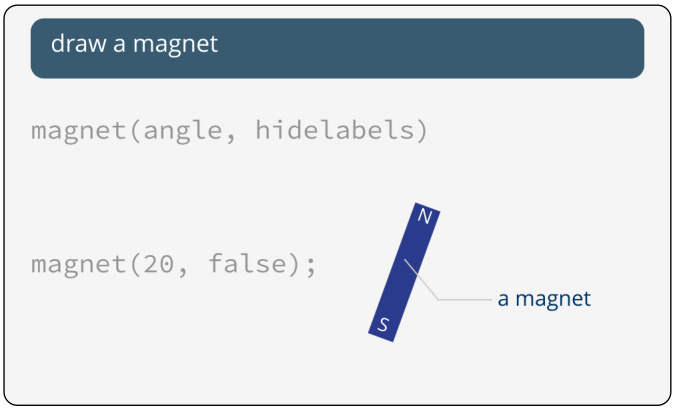
parameters

`angle`: rotation of the bar magnet, in degrees

`hidelabels`: a boolean - true or false to show or hide labels.

examples

```
magnet(50, false);
magnet(20, true);
```



coil

primitive and parameters

`coil(angle)`

description

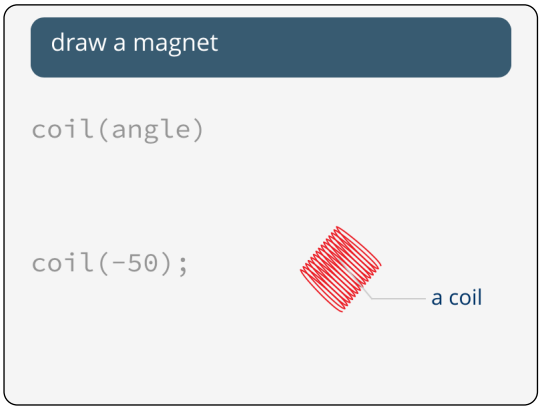
Draws a coil, rotated by the chosen angle, centred on the middle of the axis of the coil.

parameters

`angle`: rotation of the coil, in degrees

examples

```
coil(20);
coil(-50);
```



fatcoil

primitive and parameters

`fatcoil()`

description

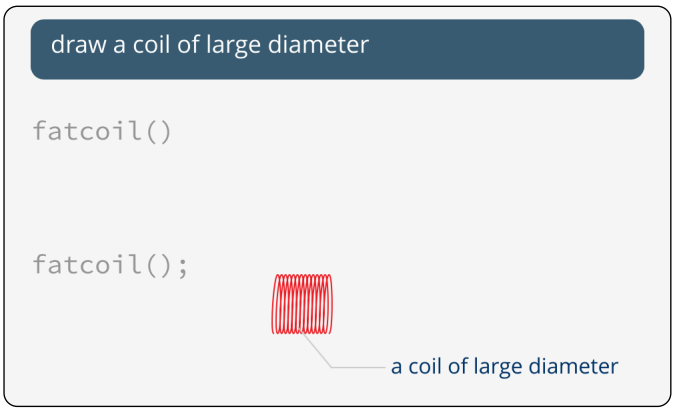
Draws a coil of large diameter.

parameters

none

examples

```
fatcoil();
```



thincoil

primitive and parameters

`thincoil()`

description

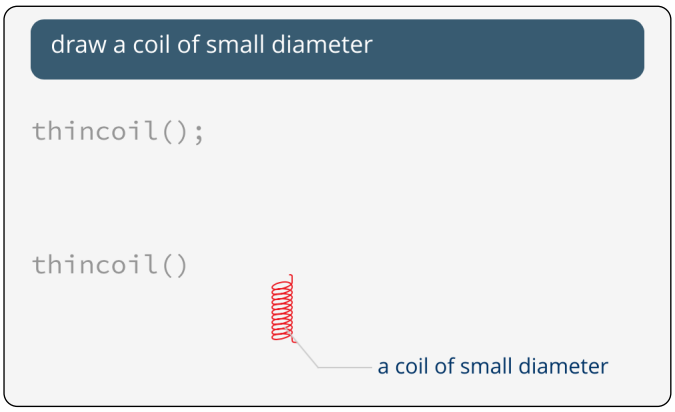
Draws a coil of small diameter.

parameters

none

examples

```
thincoil();
```



compass

primitive and parameters

`compass(angle)`

description

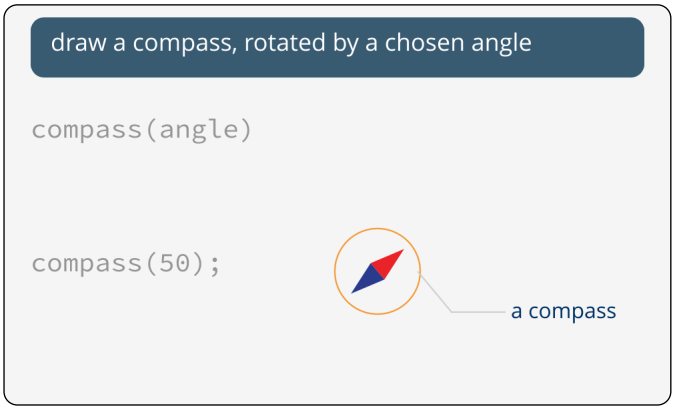
Draws a compass, rotated at the chosen angle in degrees.

parameters

`angle`: the angle of rotation of the compass in degrees

examples

```
compass(-20);
compass(50);
```



PoV

primitive and parameters

`PoV(name)`

description

You can represent a point of view with one of three types of representation: an object and eye, an eye, or just the object. Here we deal with the PoV, which is both eye and object.

Alice takes one point of view and is always green.

Bob takes another point of view and is always brown.

Charlie takes yet another point of view, and is always purple.

All three can face either right or left.

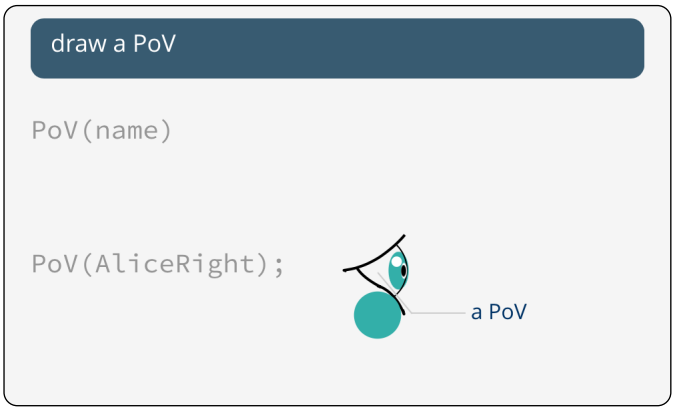
Creates a PoV with specified kind—one of: AliceLeft; AliceRight; BobLeft; BobRight; CharlieLeft; CharlieRight.

parameters

`name:` one of AliceLeft; AliceRight; BobLeft; BobRight; CharlieLeft; CharlieRight

examples

```
PoV('BobLeft');
PoV('AliceRight');
```



PoVEye

primitive and parameters

`PoVEye(name)`

description

You can represent a point of view with one of three types of representation: an object and eye, an eye, or just the object. Here we deal with the PoVEye, which is just the eye.

Alice takes one point of view and is always green.

Bob takes another point of view and is always brown.

Charlie takes yet another point of view, and is always purple.

All three can face either right or left.

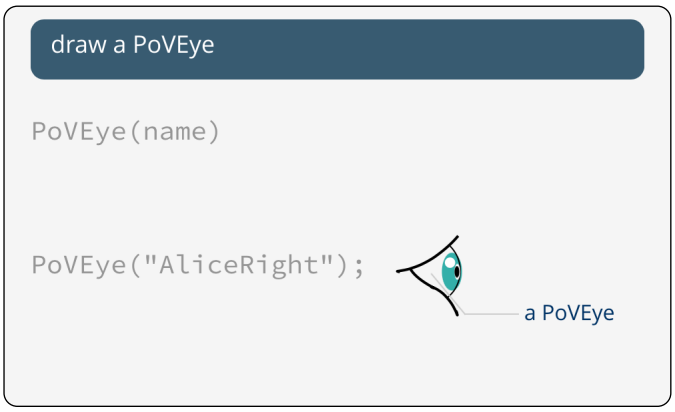
Creates a PoVEye with specified kind—one of: AliceLeft; AliceRight; BobLeft; BobRight; CharlieLeft; CharlieRight.

parameters

`name:` one of AliceLeft; AliceRight; BobLeft; BobRight; CharlieLeft; CharlieRight

examples

```
PoVEye("AliceRight");
```



PoVObject

primitive and parameters

`PoVObject(name)`

description

You can represent a point of view with one of three types of representation: an object and eye, an eye, or just the object. Here we deal with the PoVObject, which is just the object.

Alice takes one point of view and is always green.

Bob takes another point of view and is always brown.

Charlie takes yet another point of view, and is always purple.

All three can face either right or left.

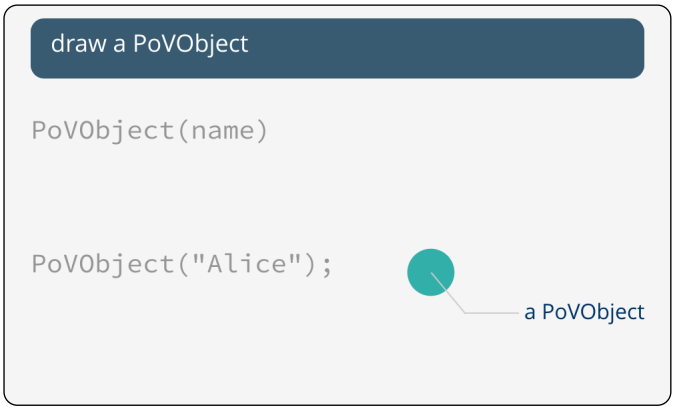
Creates a PoVObject with specified kind—one of: Alice, Bob, Charlie.

parameters

`name:` one of Alice, Bob, Charlie

examples

```
PoVObject("Alice");
```



device

primitive and parameters

`device(number)`

description

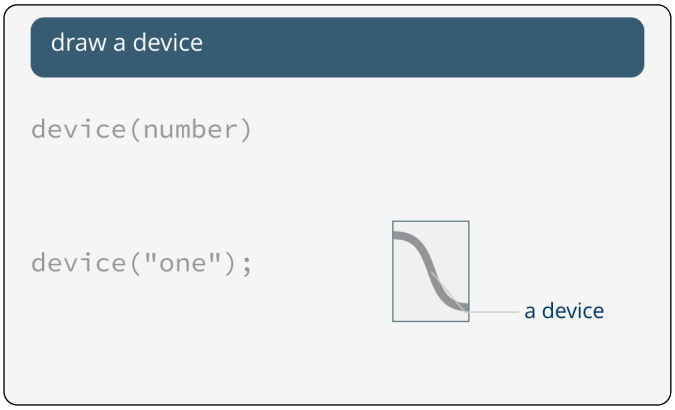
Creates a device with specified kind—one of: one; two; three.

parameters

`number` : one, two, three

examples

```
device("two");
device("one");
```



powerPathway

primitive and parameters

`powerPathway(kind)`

description

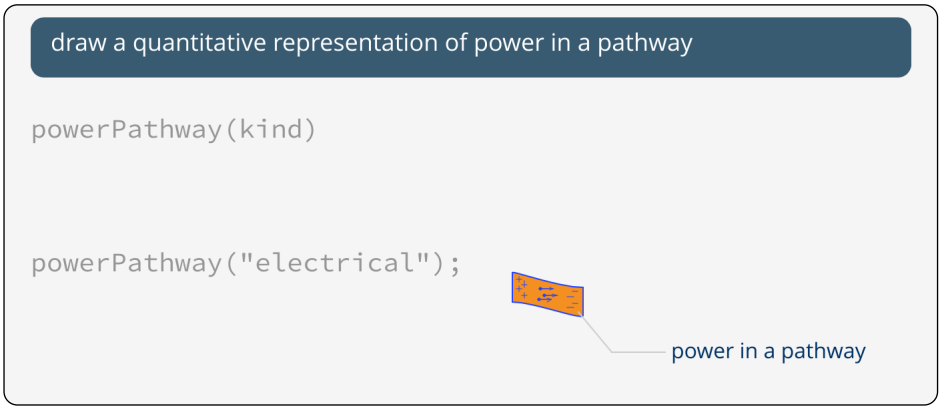
Creates a pathway with specified kind—one of: electrical; particles; mechanical; radiation.

parameters

`kind:` electrical, particles, mechanical, radiation

examples

```
powerPathway("mechanical");
powerPathway("electrical");
```



storeEmptier

primitive and parameters

`storeEmptier(type)`

description

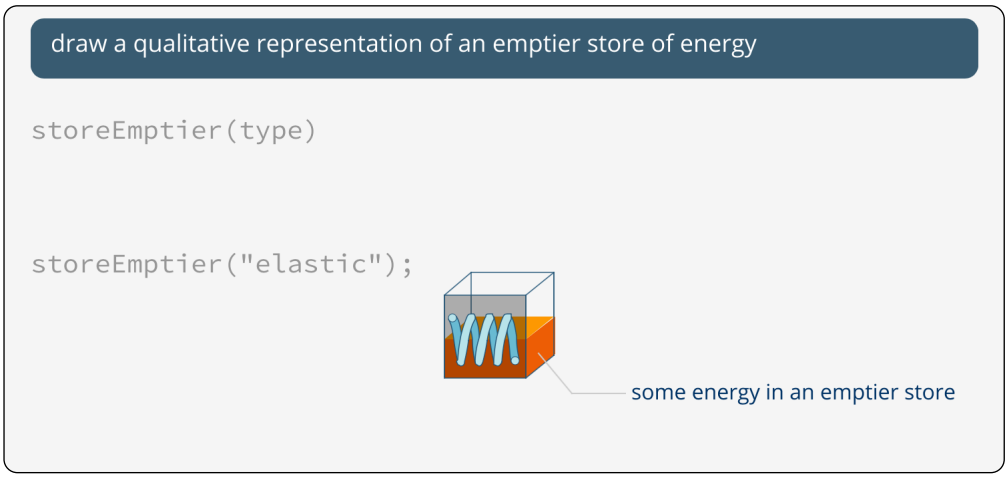
Creates a rather empty energy store with specified kind—one of: elastic; chemical; vibration; gravity; kinetic; thermal; nuclear; electromagnetic.

parameters

`type` : elastic; chemical; vibration; gravity; kinetic; thermal; nuclear; electromagnetic

examples

```
storeEmptier("gravity");
storeEmptier("elastic");
```



storeFuller

primitive and parameters

`storeFuller(type)`

description

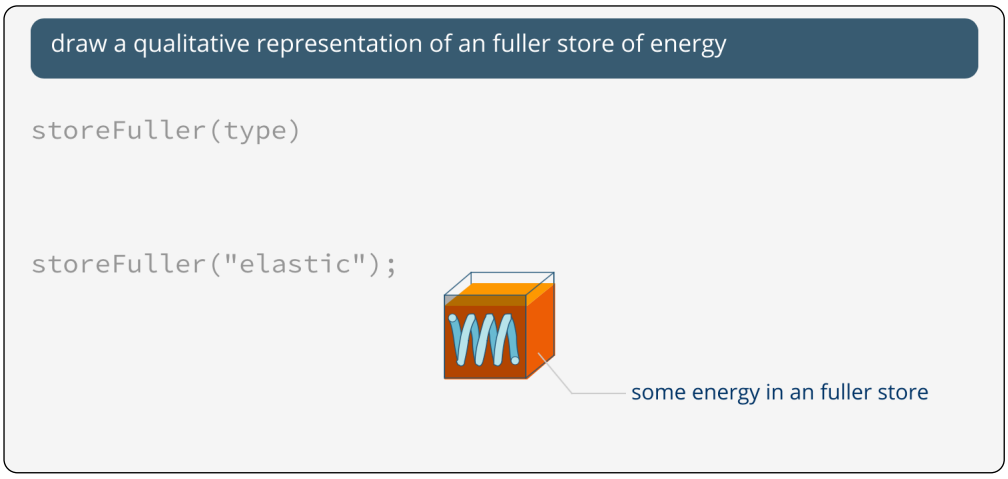
Creates a rather full energy store with specified kind—one of: elastic; chemical; vibration; gravity; kinetic; thermal; nuclear; electromagnetic.

parameters

`type` : elastic; chemical; vibration; gravity; kinetic; thermal; nuclear; electromagnetic

examples

```
storeFuller("thermal");
storeFuller("elastic");
```



physics quantitative

quantity

primitive and parameters

`quantity(value,qcolor,label)`

description

Draws a bar to represent any quantity, with an optional label.

parameters

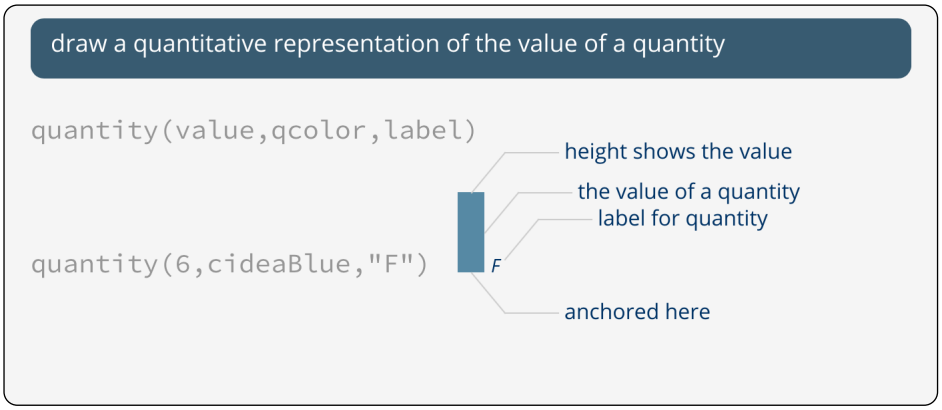
`value`:

`qcolor`:

`label`:

examples

```
quantity(6, cideaRed, "how much")
```



energy

primitive and parameters

`energy(value)`

description

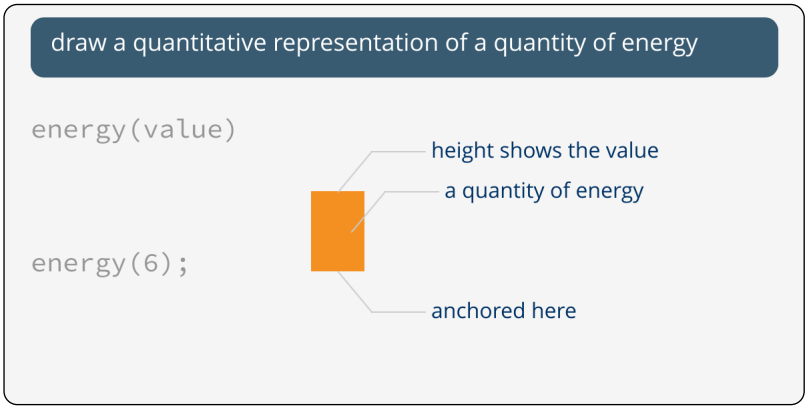
Draws an orange bar, representing a quantity of energy.

parameters

`value`: the quantity of energy to be represented, which must be positive

examples

```
energy(6);
```



power

primitive and parameters

`power(value)`

description

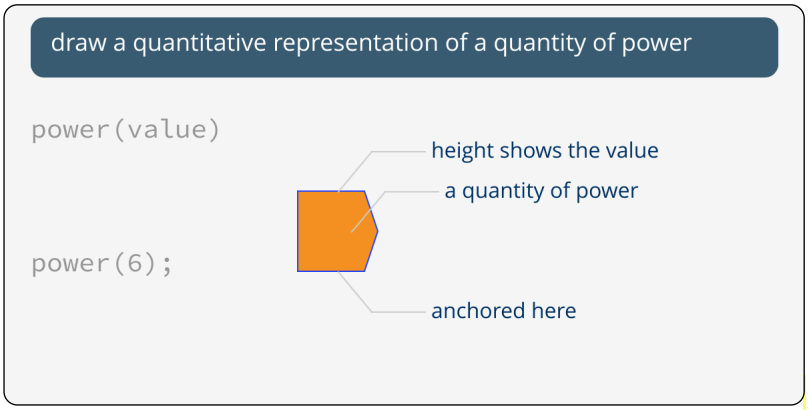
Draws a Sankey arrow, representing a quantity of power.

parameters

`value`: the quantity of power, which can be positive or negative

examples

```
power(-10);
power(6);
```



quantum

primitive and parameters

`quantum(value)`

description

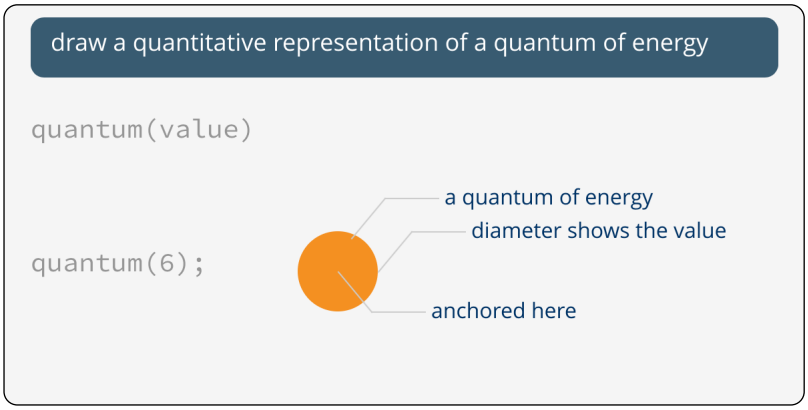
Draws an orange circle, representing the value of the energy shifted by the quantum.

parameters

`value`: the value of the energy shifted by the quantum, which must be positive

examples

```
quantum(2);
quantum(6);
```



resistance

primitive and parameters

`resistance(value)`

description

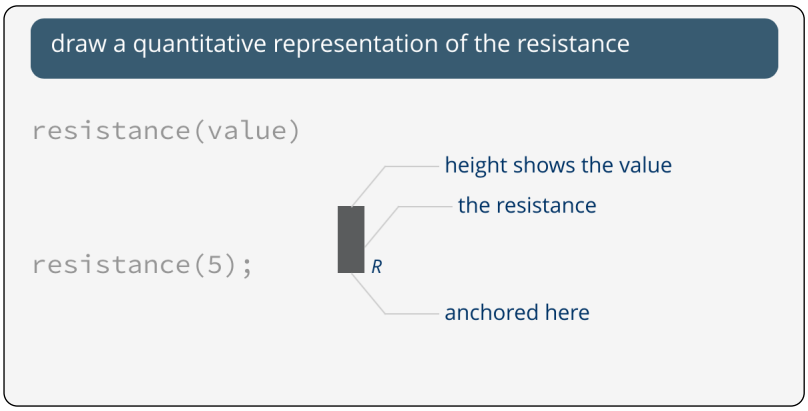
Draws a bar the represent the value of the resistance. Offset by 110 pixels to the right, so that it can share an origin with a circuit element easily.

parameters

`value`: the value of the resistance, which must be positive.

examples

```
resistance(2);
resistance(5);
```



current

primitive and parameters

`current(value)`

description

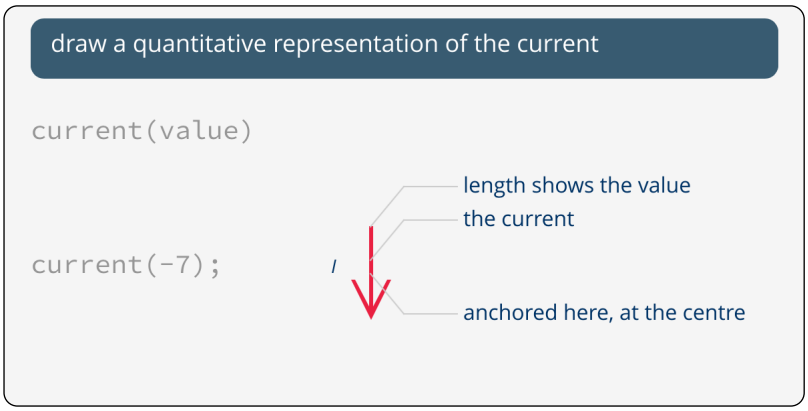
Draws an arrow to represent the current, offset by 76 pixels to the left, so that it can share an origin with a circuit element easily.

parameters

`value`: the value of the current, which can be positive or negative

examples

```
current(4);
current(-7);
```



current3D

primitive and parameters

`current3D(value)`

description

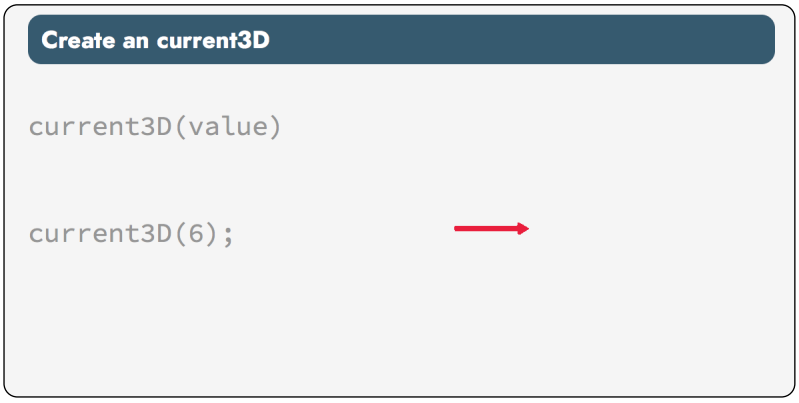
Draws an arrow to represent the current, with the zero constant.

parameters

`value`: the value of the current, which can be positive or negative

examples

```
current(6);
current(-6);
```



pd

primitive and parameters

`pd(value)`

description

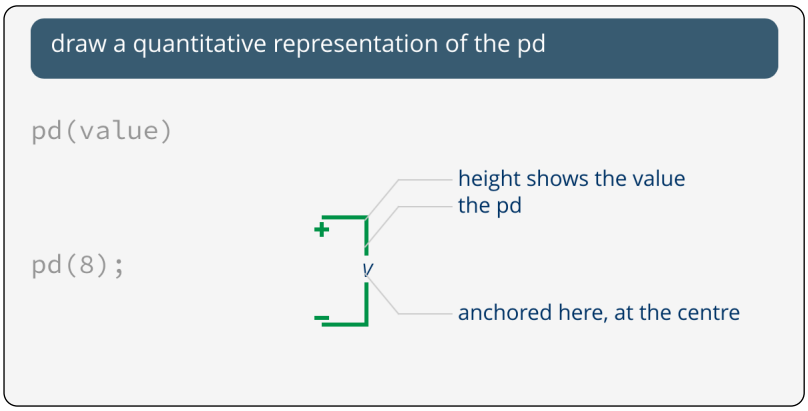
Draws an stylised connected voltmeter to represent the pd, offset by 70 pixels to the right, so that it can share an origin with a circuit element easily.

parameters

`value`: the value of the pd, which can be positive or negative

examples

```
pd(-5);
pd(8);
```



ray

primitive and parameters

`ray(fromx,fromy,magnitude,angle)`

description

parameters

`fromx`: start point

`fromy`: start point

`magnitude`: length of ray

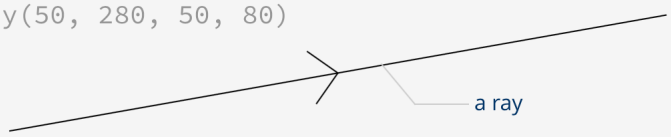
`angle`: angle of ray

examples

draw ray, choosing origin, magnitude and direction

```
ray(fromx,fromy,magnitude,angle)


ray(50, 280, 50, 80)
```



rayC

primitive and parameters

`rayC(fromx,fromy,tox,toy)`

description

parameters

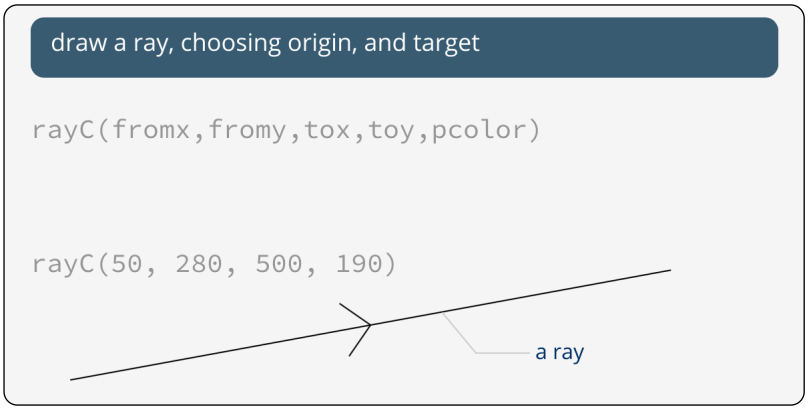
`fromx`: start point

`fromy`: start point

`tox`: end point

`toy`: end point

examples



path

primitive and parameters

`path(fromx,fromy,magnitude,angle,pcolor)`

description

Draws a path for reasoning about contributions, of specified colour.

parameters

`fromx`: start point

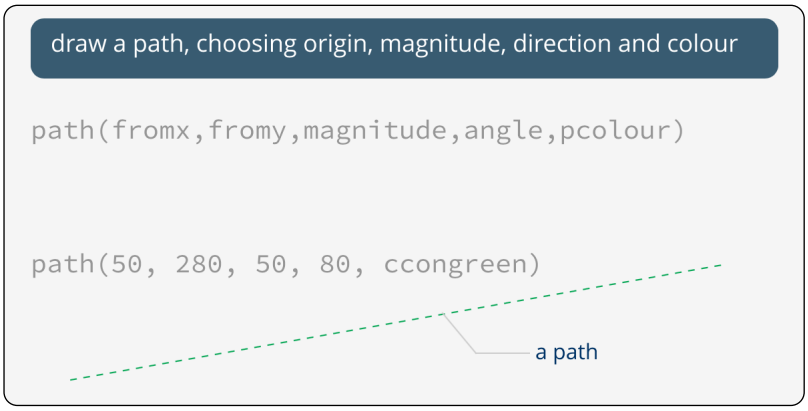
`fromy`: start point

`magnitude`: length of path

`angle`: angle of path

`pcolour`: colour of the path

examples



pathC

primitive and parameters

`pathC(fromx,fromy,tox,toy,pcolor)`

description

Draws a path for reasoning about contributions, of specified colour.

parameters

`fromx`: start point

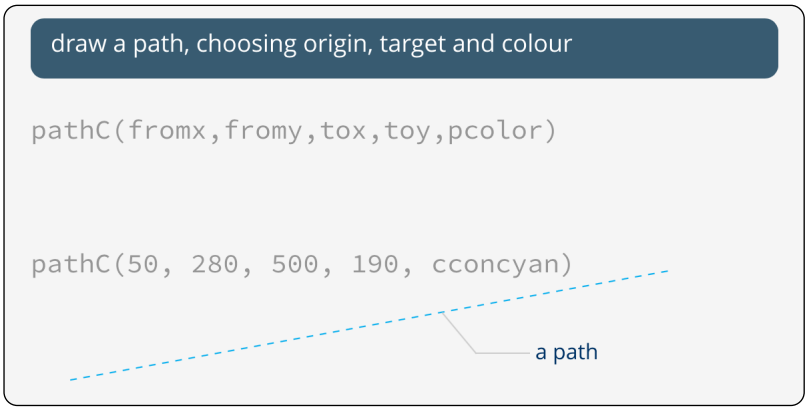
`fromy`: start point

`tox`: end point

`toy`: end point

`pcolor`: path colour

examples



pathC3D

primitive and parameters

`pathC3D(fromx,fromy,tox,toy,pcolor)`

description

Draws a path for reasoning about contributions, of specified colour, suitable for 3D rendering.

parameters

`fromx`: start point

`fromy`: start point

`tox`: end point

`toy`: end point

`pcolor`: path colour

examples

```
pathC3D(-100,100,250,100, ccongreen);
```

Create an pathC3D

```
pathC3D(fromx,fromy,tox,toy,pcolor)


pathC3D(-100,100,250,100, ccongreen);


                ---------------------------------
```

beam

primitive and parameters

`beam(fromx,fromy,magnitude,angle,bcolor)`

description

Draws a beam of radiation, of specified colour.

parameters

`fromx`: start point

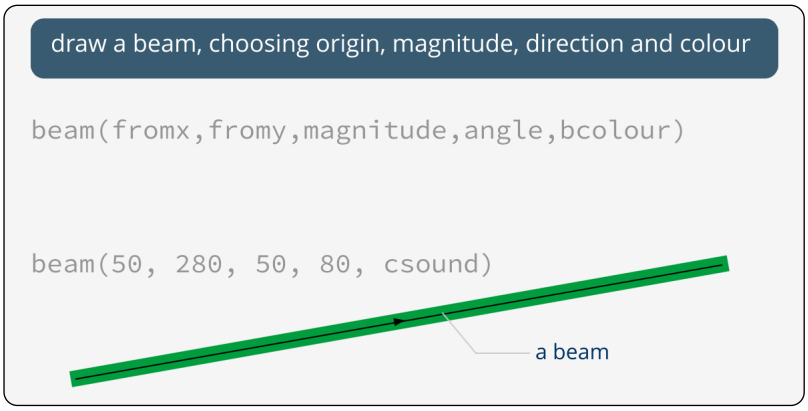
`fromy`: start point

`magnitude`: length of path

`angle`: angle of path

`bcolour`: colour of the beam

examples



beamC

primitive and parameters

`beamC(fromx,fromy,tox,toy,bcolor)`

description

Draws a beam of radiation, of specified colour.

parameters

`fromx`: start point

`fromy`: start point

`tox`: end point

`toy`: end point

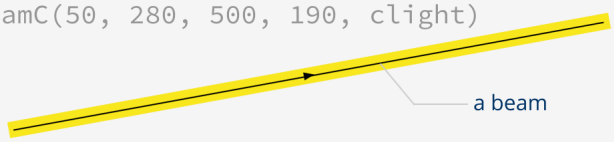
`bcolor`: path colour

examples

draw a beam, choosing origin, target and colour

```
beamC(fromx,fromy,tox,toy,pcolor)


beamC(50, 280, 500, 190, clight)
```



amplitude

primitive and parameters

`amplitude(magnitude,angle,acolour)`

description

Draws a stylised arrow, representing an amplitude.

parameters

`magnitude`: magnitude of amplitude

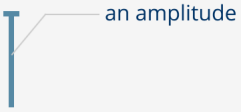
`angle`: rotation of amplitude

`acolour`: colour of amplitude

examples

draw an amplitude, choosing magnitude, angle and colour

```
amplitude(magnitude,angle,acolour)
```




```
amplitude(7, 0, cideaBlue)
```

contribution

primitive and parameters

`contribution(magnitude,angle,ccolor)`

description

Draws a stylised arrow, representing a contribution (used in reasoning about superposition).

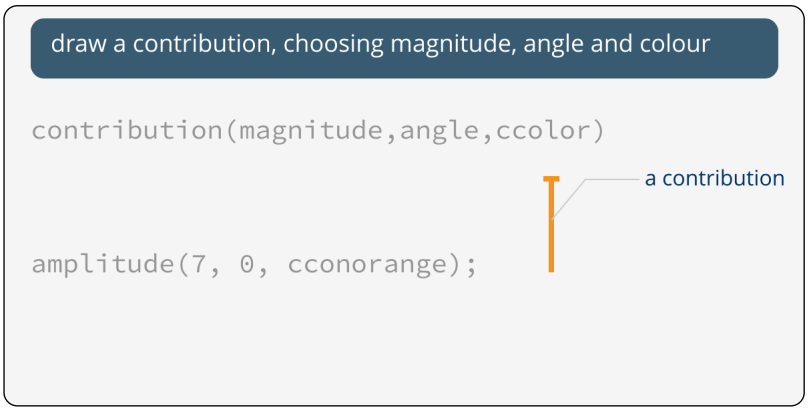
parameters

`magnitude:` magnitude of contribution, grows from bottom

`angle`: rotation of contribution, 0 is straight up, in degrees

`acolour`: colour of contribution

examples



phasorArrow

primitive and parameters

`phasorArrow(amplitude,angularfrequency,pathtime,phasorcolour)`

description

Draws a phasor, representing a contribution (used in reasoning about superposition and paths).

parameters

`amplitude:` amplitude of phasor: magnitude of arrow

`angularfrequency`: rate of rotation of phasor, in radians

`pathtime:` duration of path between emission and detection

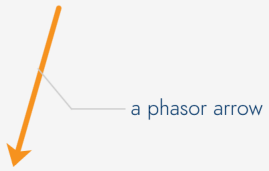
`phasorcolour`: colour of phasor arrow

examples

draw a phasorArrow, of chosen amplitude, angular frequency, path time, and colour

```
phasorArrow(amplitude,angularfrequency,pathtime,phasorcolour)


phasorArrow(12, 0.5, 3.2, cconorange);
```



phasormultiple

primitive and parameters

`phasormultiple(amplitude,angularfrequency,phasordata)`

description

Draws a set of phasors.

parameters

`amplitude:` amplitude of phasor: magnitude of arrow

`angularfrequency`: rate of rotation of phasor, in radians

`phasordata:` an array of pathtimes and colours

examples

draw a phasormultiple, choosing amplitude, angular frequency, & phasor data

```
phasormultiple(amplitude,angularfrequency,phasordata)


phasormultiple(4, 0.3, [[2,cconcyan],
              [3,cconlightgreen],[2,cconpink]])
```

phasormultipleresultant

primitive and parameters

`phasormultipleresultant(amplitude,angularfrequency,phasordata)`

description

Draws a set of phasors and the resultant.

parameters

`amplitude:` amplitude of phasor: magnitude of arrow

`angularfrequency`: rate of rotation of phasor, in radians

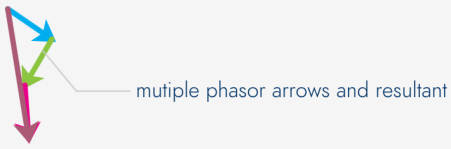
`phasordata:` an array of pathtimes and colours

examples

draw a phasormultipleresultant, choosing amplitude, angular frequency, & phasor data

```
phasormultipleresultant(amplitude,angularfrequency,phasordata)


phasormultipleresultant(4, 0.3, [[2,cconcyan],
              [7,cconlightgreen],[5,cconpink]])
```



phasormultipleresultantbrightness

primitive and parameters

`phasormultipleresultantbrightness(amplitude,angularfrequency,phasordata)`

description

Draws a set of phasors, the resultant and the intensity.

parameters

`amplitude:` amplitude of phasor: magnitude of arrow

`angularfrequency`: rate of rotation of phasor, in radians

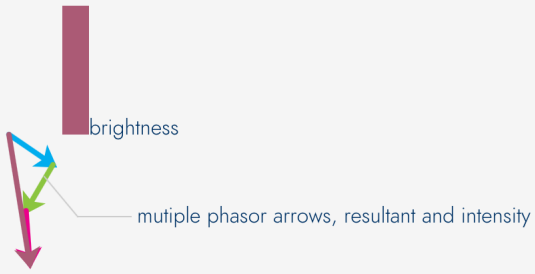
`phasordata:` an array of pathtimes and colours

examples

draw a phasormultipleresultantbrightness, choosing amplitude, angular frequency, & phasor data

```
phasormultipleresultantbrightness(amplitude,angularfrequency,
       phasordata)

phasormultipleresultantbrightness(4, 0.3, [[2,cconcyan],
              [7,cconlightgreen],[5,cconpink]])
```



phasormultiplebrightness

primitive and parameters

`phasormultiplebrightness(amplitude,angularfrequency,phasorda-ta)`

description

Draws a set of phasors and the resultant intensity.

parameters

`amplitude:` amplitude of phasor: magnitude of arrow

`angularfrequency`: rate of rotation of phasor, in radians

`phasordata:` an array of pathtimes and colours

examples

draw a phasormultiplebrightness, choosing amplitude, angular frequency, & phasor data

```
phasormultiplebrightness(amplitude,angularfrequency,
       phasordata)

phasormultiplebrightness(4, 0.3, [[2,cconcyan],
              [7,cconlightgreen],[5,cconpink]])
```

mass

primitive and parameters

`mass(value,mcolour)`

description

Draws a square to represent a mass, of specified colour.

parameters

`value`: quantity of mass, shown by width or height, grows from centre

`mcolour`: colour of mass

examples

draw a mass, choosing value and colour

```
mass(value,mcolour)


mass(4, cideaGrey)
```

duration

primitive and parameters

`duration(value,maxValue)`

description

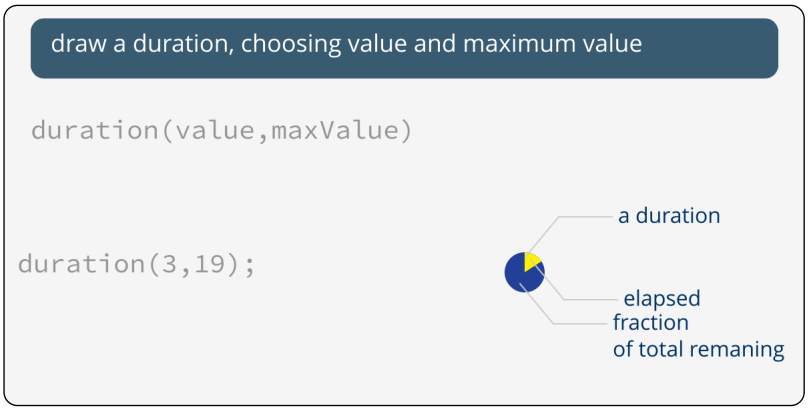
Draws a partially filled circle, too represent a duration.

parameters

`value`: of duration

`maxValue`: of duration

examples



durationpov

primitive and parameters

`durationpov(value,maxValue,durationpovcolour)`

description

Draws a partially filled circle of specified colour, too represent a duration

parameters

`value`: of duration

`maxValue`: of duration

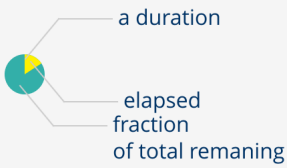
`durationpovcolour`: colour of duration

examples

draw a coloured duration, choosing value and maximum value

```
durationpov(value,maxValue,durationpovcolour)
```

```
durationpov(3,19,cpovAlice);
```



timeclock

primitive and parameters

`timeclock(value,maxValue)`

description

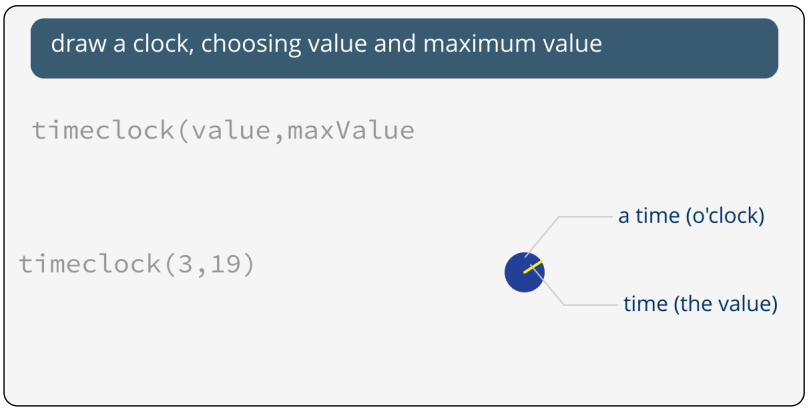
Draws a circle, with a radius to represent a time on the clock (a particular instant).

parameters

`value`: shown on timeclock

`maxValue`: shown by timeclock

examples



timepovclock

primitive and parameters

`timepovclock(value,maxValue,timepovcolour)`

description

Draws a circle of specified colour, with a radius to represent a time on the clock (a particular instant)

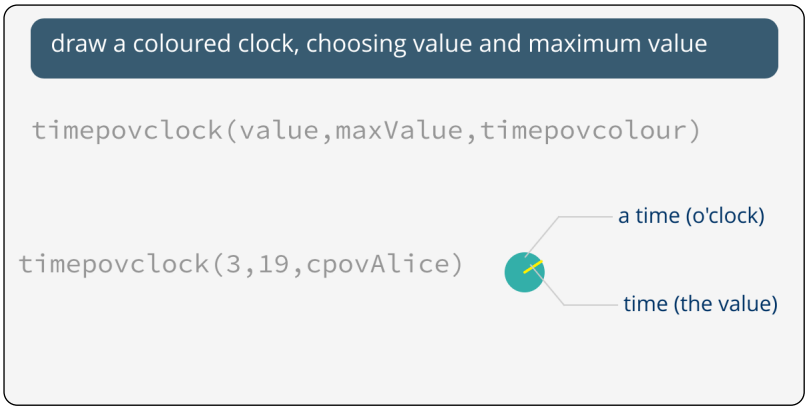
parameters

`value`: of time shown

`maxValue`: to be shown by clock after one complete rotation

`timepovcolour`: of clock

examples



displacement

primitive and parameters

`displacement(magnitude,rotation,dcolour)`

description

Draws a stylised arrow, representing a displacement.

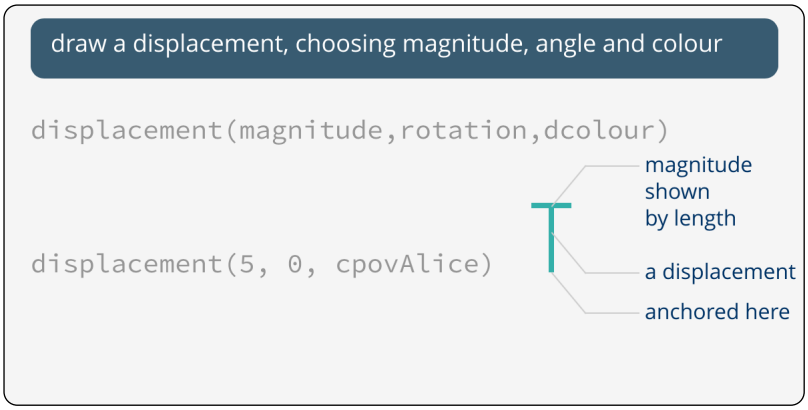
parameters

`magnitude`: magnitude of displacement

`rotation`: rotation of displacement

`dcolour`: colour of displacement

examples



displacement3D

primitive and parameters

`displacement3D(magnitude,rotation,dcolour)`

description

Draws a stylised arrow, representing a displacement, suitable for 3D rendering.

parameters

`magnitude`: magnitude of displacement

`rotation`: rotation of displacement

`dcolour`: colour of displacement

examples

```
displacement3D(6, 90, cpovAlice);
```

Create an displacement3D

```
displacement3D(magnitude,rotation,dcolour)

displacement3D(6, 90, cpovAlice);
```

velocity

primitive and parameters

`velocity(magnitude,rotation,vcolour)`

description

Draws a stylised arrow, representing a velocity.

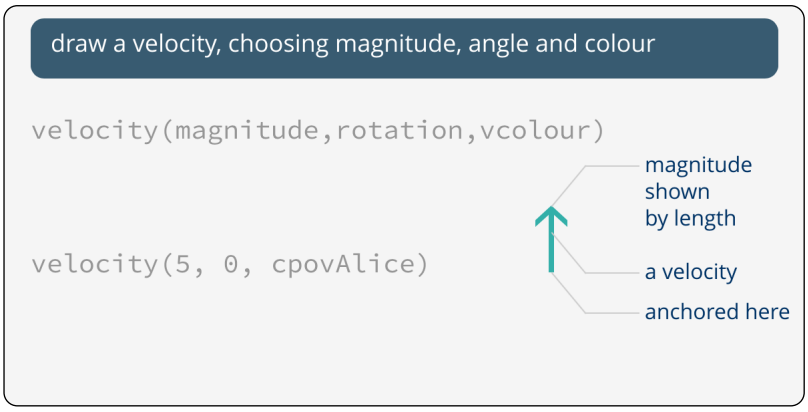
parameters

`magnitude`: magnitude of velocity

`rotation`: rotation of velocity

`vcolour`: colour of velocity

examples



velocity3D

primitive and parameters

`velocity3D(magnitude,rotation,vcolour)`

description

Draws a stylised arrow, representing a velocity, suitable for 3D rendering.

parameters

`magnitude`: magnitude of velocity

`rotation`: rotation of velocity

`vcolour`: colour of velocity

examples

```
velocity3D(6, 90, cpovAlice);
```

Create a velocity3D

```
velocity3D(magnitude,rotation,vcolour)

velocity3D(6, 90, cpovAlice);
```

acceleration

primitive and parameters

`acceleration(magnitude,rotation,acolour)`

description

Draws a stylised arrow, representing an acceleration.

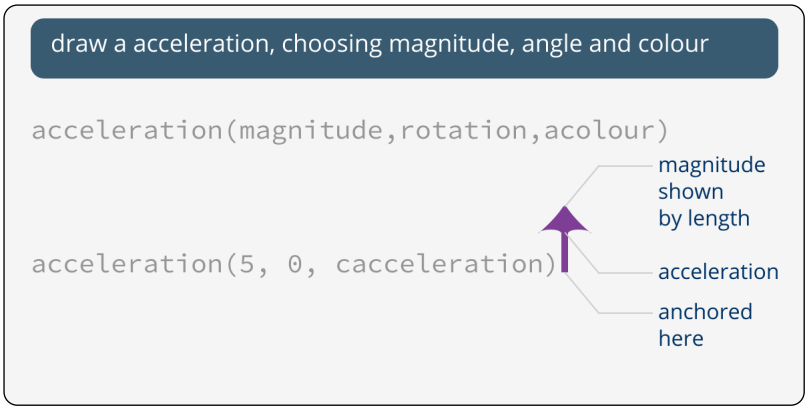
parameters

`magnitude`: magnitude of acceleration

`rotation`: rotation of acceleration

`acolour`: colour of acceleration

examples



acceleration3D

primitive and parameters

`acceleration3D(magnitude,rotation,acolour)`

description

Draws a stylised arrow, representing an acceleration, suitable for 3D rendering.

parameters

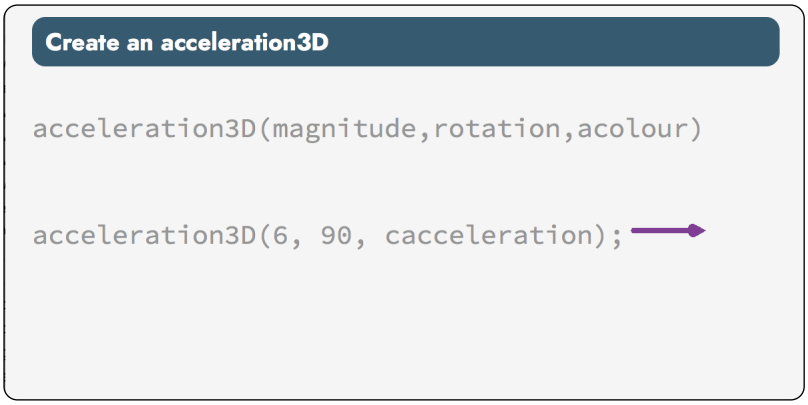
`magnitude`: magnitude of acceleration

`rotation`: rotation of acceleration

`acolour`: colour of acceleration

examples

```
acceleration3D(6, 90, cacceleration);
```



momentum

primitive and parameters

`momentum(magnitude,rotation,momcolour)`

description

Draws a stylised arrow, representing a momentum.

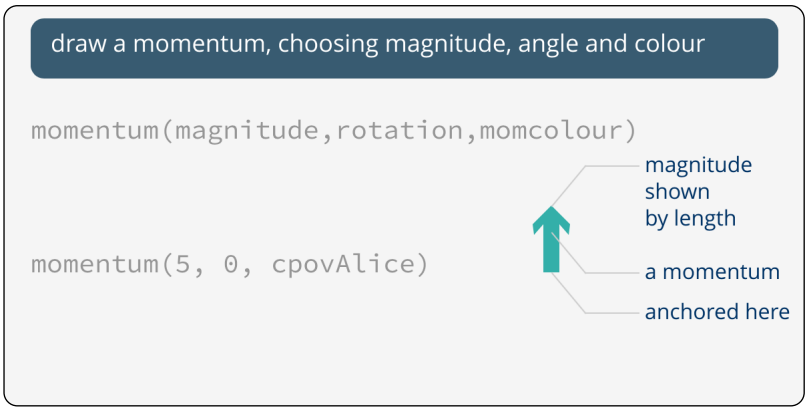
parameters

`magnitude`: magnitude of momentum

`rotation`: rotation of momentum

`momcolour`: colour of momentum

examples



force

primitive and parameters

`force(magnitude, rotation, fcolour)`

description

Draws a stylised arrow, representing a force.

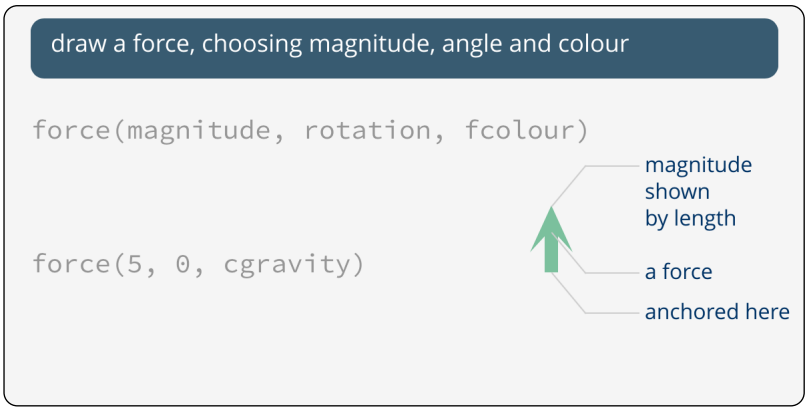
parameters

`magnitude`: magnitude of force

`rotation`: rotation of force

`fcolour`: colour of force

examples



force3D

primitive and parameters

`force3D(magnitude, rotation, fcolour)`

description

Draws a stylised arrow, representing a force, suitable for 3D rendering

parameters

`magnitude`: magnitude of force

`rotation`: rotation of force

`fcolour`: colour of force

examples

`force3D(6, 90, cgravity3D);`

Create an force3D

```
force3D(magnitude, rotation, fcolour)
```

```
force3D(6, 90, cgravity3D);
```

trackSegment

primitive and parameters

`trackSegment(fromx,fromy,tox,toy)`

description

Draws a stylised line, representing a track segment.

parameters

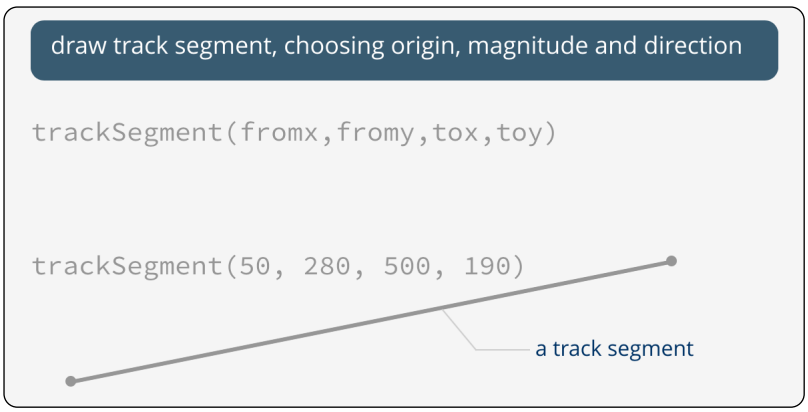
`fromx`: start point

`fromy`: start point

`tox`: end point

`toy`: end point

examples



trackSegment3D

primitive and parameters

`trackSegment3D(fromx,fromy,tox,toy)`

description

Draws a stylised line, representing a track segment, suitable for 3D rendering.

parameters

`fromx`: start point

`fromy`: start point

`tox`: end point

`toy`: end point

examples

`trackSegment3D(-100,100,250,100);`

Create an trackSegment3D

```
trackSegment3D(fromx,fromy,tox,toy,pcolor)


trackSegment3D(-100,100,250,100, ccongreen);
```

physics compound

pulleyset

primitive and parameters

`pulleyset(numberofpulleys, separation)`

description

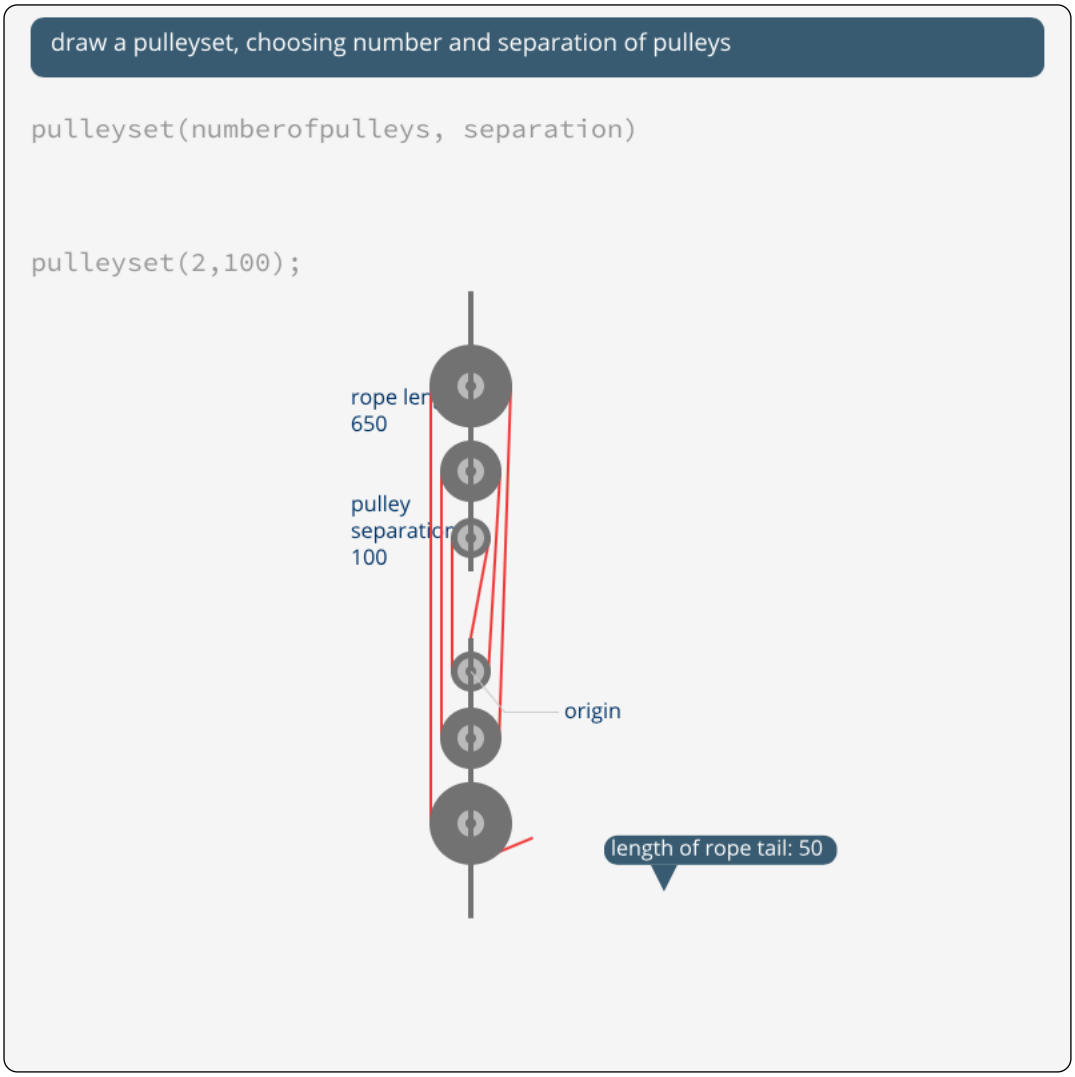
Draws a set of pulleys

parameters

`numberofpulleys`

`separation`

examples



pulley

primitive and parameters

`pulley(diameter,strapdirection)`

description

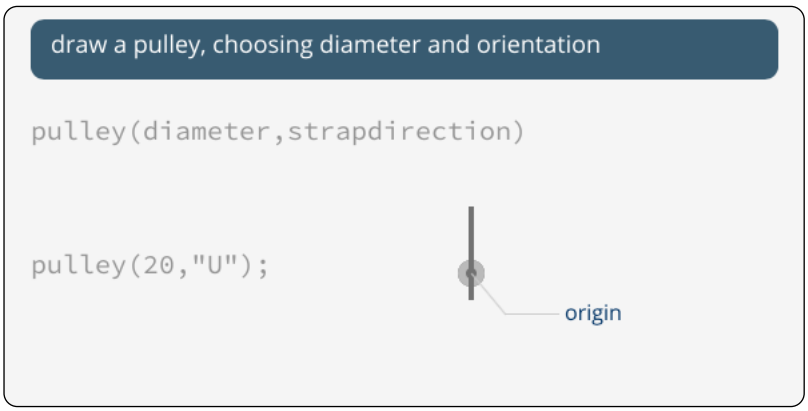
Draws a pulley and strap, which can be one of "U","D","L" or "R", to denote direction.

parameters

`diameter:` a number, to set the pulley diameter

`strapdirection`: "U","D","L" or "R"

examples



hydraulicsystem

primitive and parameters

`hydraulicsystem(ratio,volumeshiftedLR)`

description

parameters

`ratio`: of area of tubes

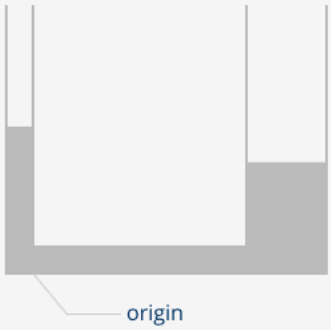
`volumeshiftedLR`: volume of fluid shifted

examples

draw a hydraulicsystem, choosing ratio and volume of fluid shifted

```
hydraulicsystem(ratio,volumeshiftedLR)
```

```
hydraulicsystem(3,400);
```



leversystem

primitive and parameters

`leversystem(multiplier, leftDisplacement)`

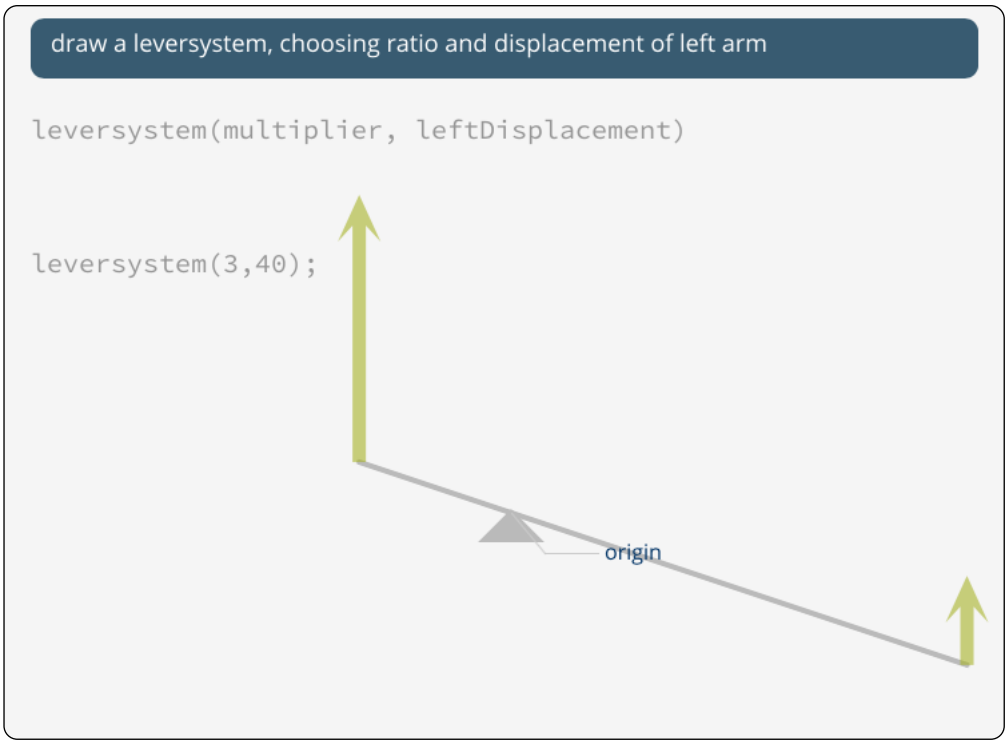
description

parameters

`multiplier`: ratio of lengths of arms

`leftDisplacement`: upward displacement of left arm

examples



pressureparticles

primitive and parameters

`pressureparticles(pressure)`

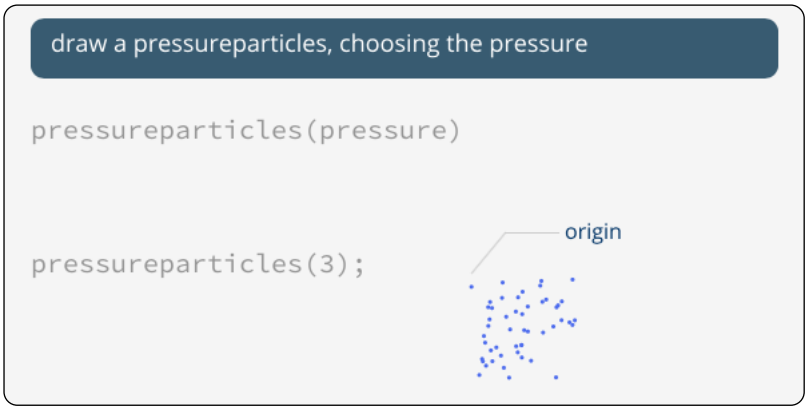
description

Draws a representation of particles at different densities

parameters

`pressure`: 0,1,2,3, or 4

examples



pressuregauge

primitive and parameters

`pressuregauge(pressure)`

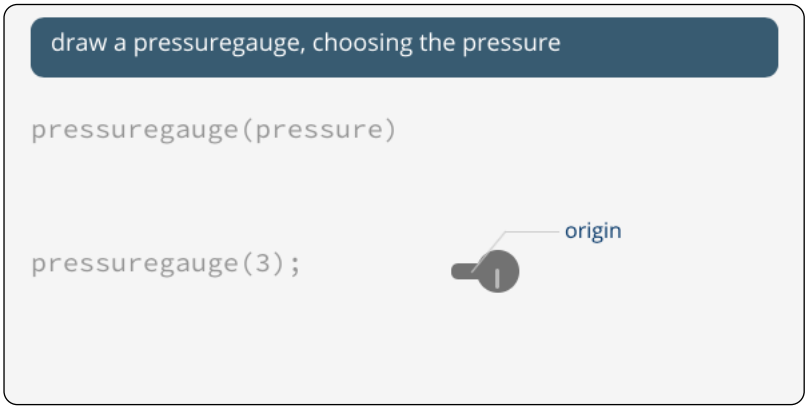
description

Draws a pressure gauge showing the pressure as set.

parameters

`pressure`: 0- 4

examples



temperaturegauge

primitive and parameters

`temperaturegauge(temperature)`

description

Draws a temperature gauge showing the temperature as set.

parameters

`temperature`: 0- 4

examples

draw a temperaturegauge, choosing the temperature

```
temperaturegauge(temperature)
```

```
temperaturegauge(3);
```

soundSource

primitive and parameters

`soundSource(amplitude,frequency)`

description

Draws a vibrating sound source, of chosen amplitude and frequency.

parameters

`amplitude`: a number to set the value

`frequency`: a number to set the value

examples

draw a soundSource, choosing the amplitude and frequency

```
soundSource(amplitude,frequency)

soundSource(12,300);
```

doLikeMe

primitive and parameters

`doLikeMe(kind,frequency,amplitude,delay)`

description

Draws a wave

parameters

`kind`: "L" or "T"

`frequency`: a number to set the value

`amplitude`: a number to set the value

`delay`: a number to set the value

examples

draw a doLikeMe, choosing the kind, amplitude and frequency and propagation delay

```
doLikeMe(kind,frequency,amplitude,delay)


doLikeMe("L", 120, 5,100);
```

storesShiftingBlank

primitive and parameters

`storesShiftingBlank()`

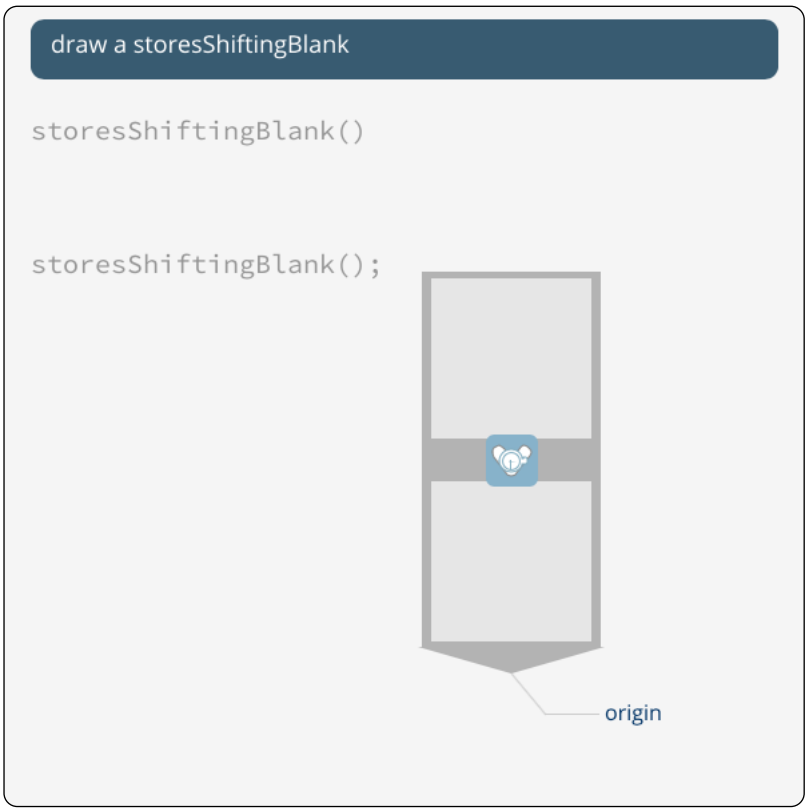
description

Draws a blank, to be filled in.

parameters

none

examples



storesAccumulatedBlank

primitive and parameters

`storesAccumulatedBlank()`

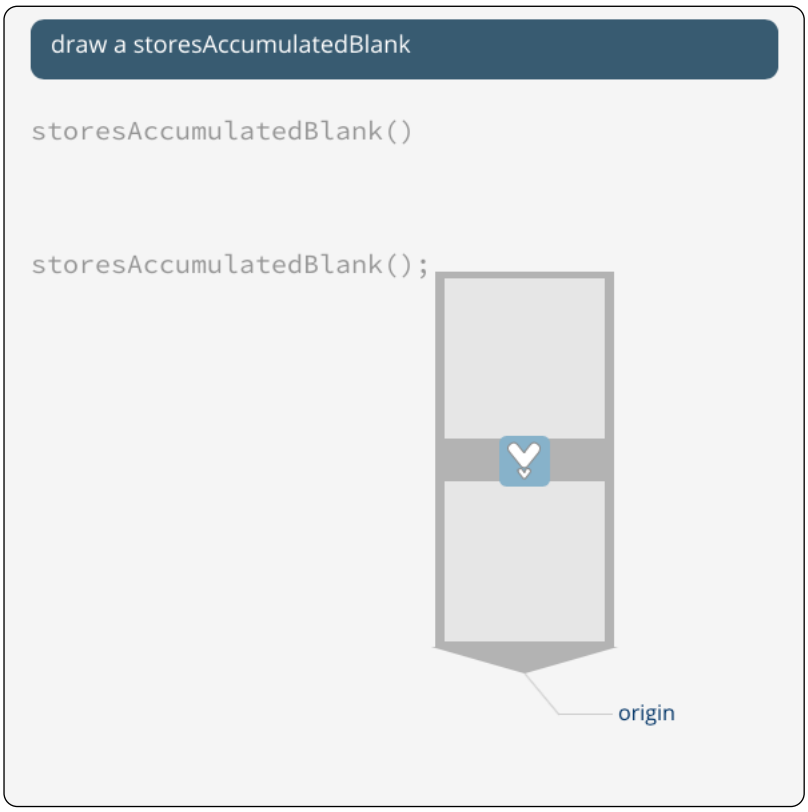
description

Draws a blank, to be filled in.

parameters

none

examples



storeFilled

primitive and parameters

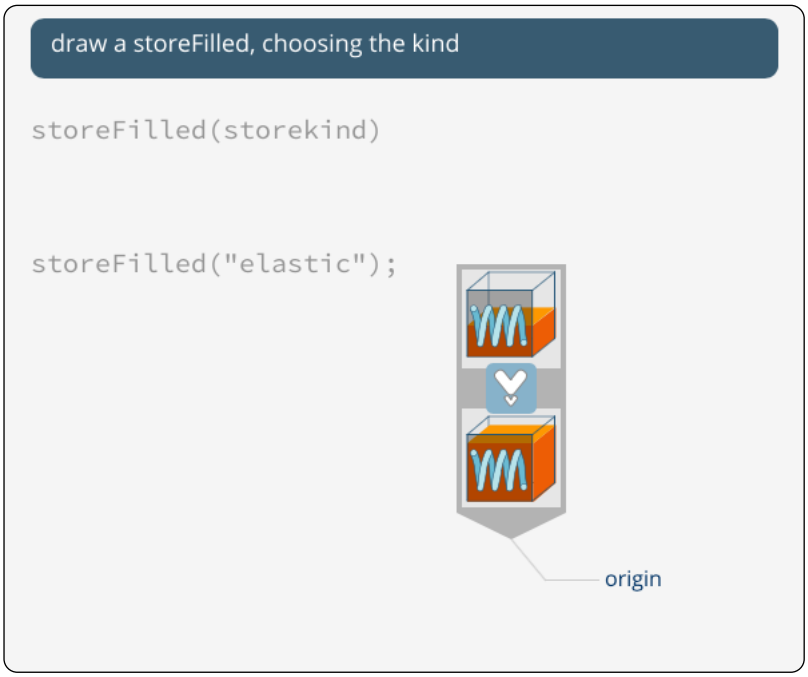
`storeFilled(storekind)`

description

parameters

`storekind`: one of "

examples



storeEmptied

primitive and parameters

`storeEmptied(storekind)`

description

parameters

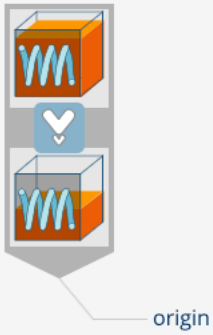
`storekind`

examples

draw a storeEmptied, choosing the kind

```
storeEmptied(storekind)
```

```
storeEmptied("elastic");
```



storeFilling

primitive and parameters

`storeFilling(storekind, pathwayKind)`

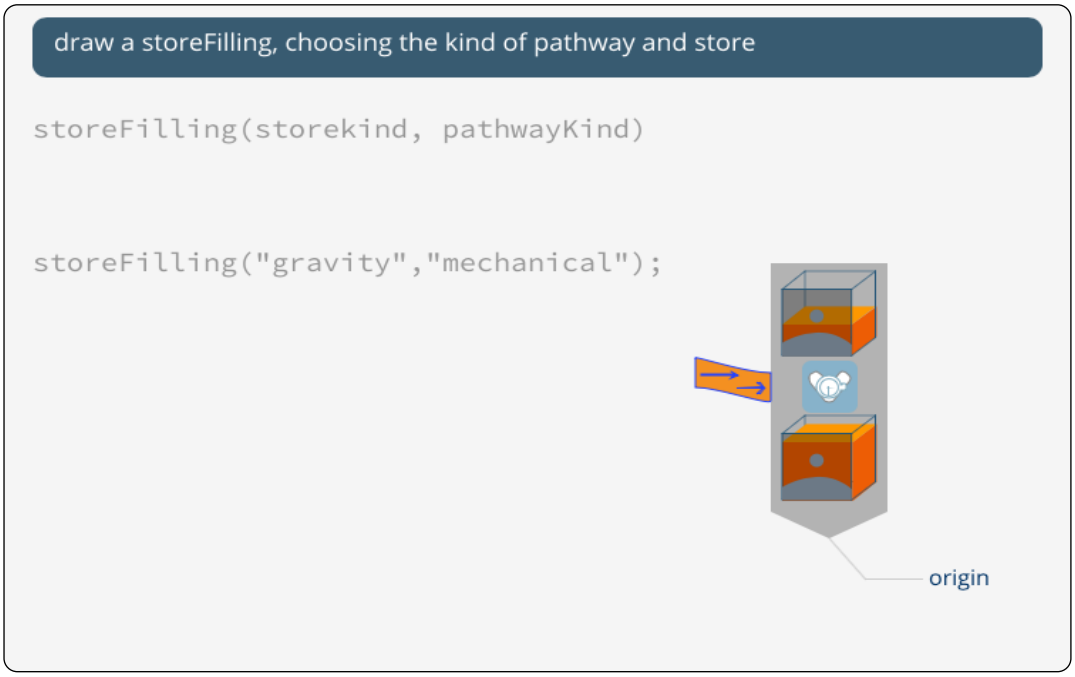
description

parameters

`storekind`

`pathwayKind`

examples



storeEmptying

primitive and parameters

`storeEmptying(storekind, pathwayKind)`

description

parameters

`storekind`

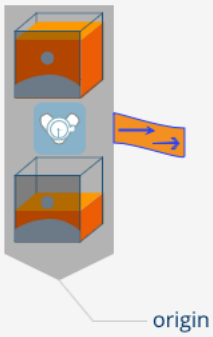
`pathwayKind`

examples

draw a storeEmptying, choosing the kind of pathway and store

```
storeEmptying(storekind, pathwayKind)


storeEmptying("gravity","mechanical");
```



describeEnergyOne

primitive and parameters

`describeEnergyOne(energyIn,energy1Out)`

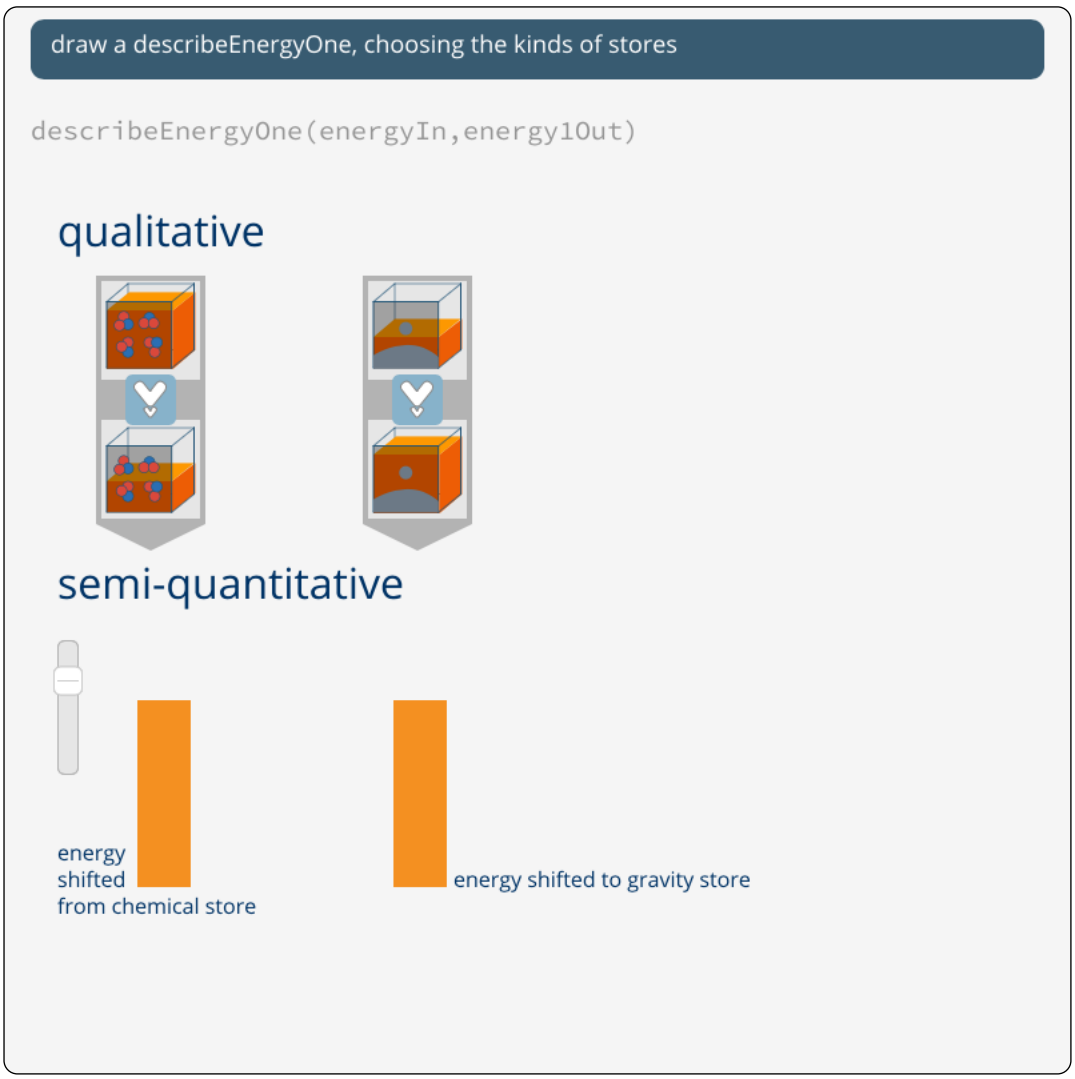
description

Requires a lot of supporting code- cf examples of use.

parameters

As above: a slider is often used to set the input, and this slider must be coded separately.

examples



describeEnergyTwo

primitive and parameters

`describeEnergyTwo(energyIn,energy1Out,energy2Out,topslice,bottomslice)`

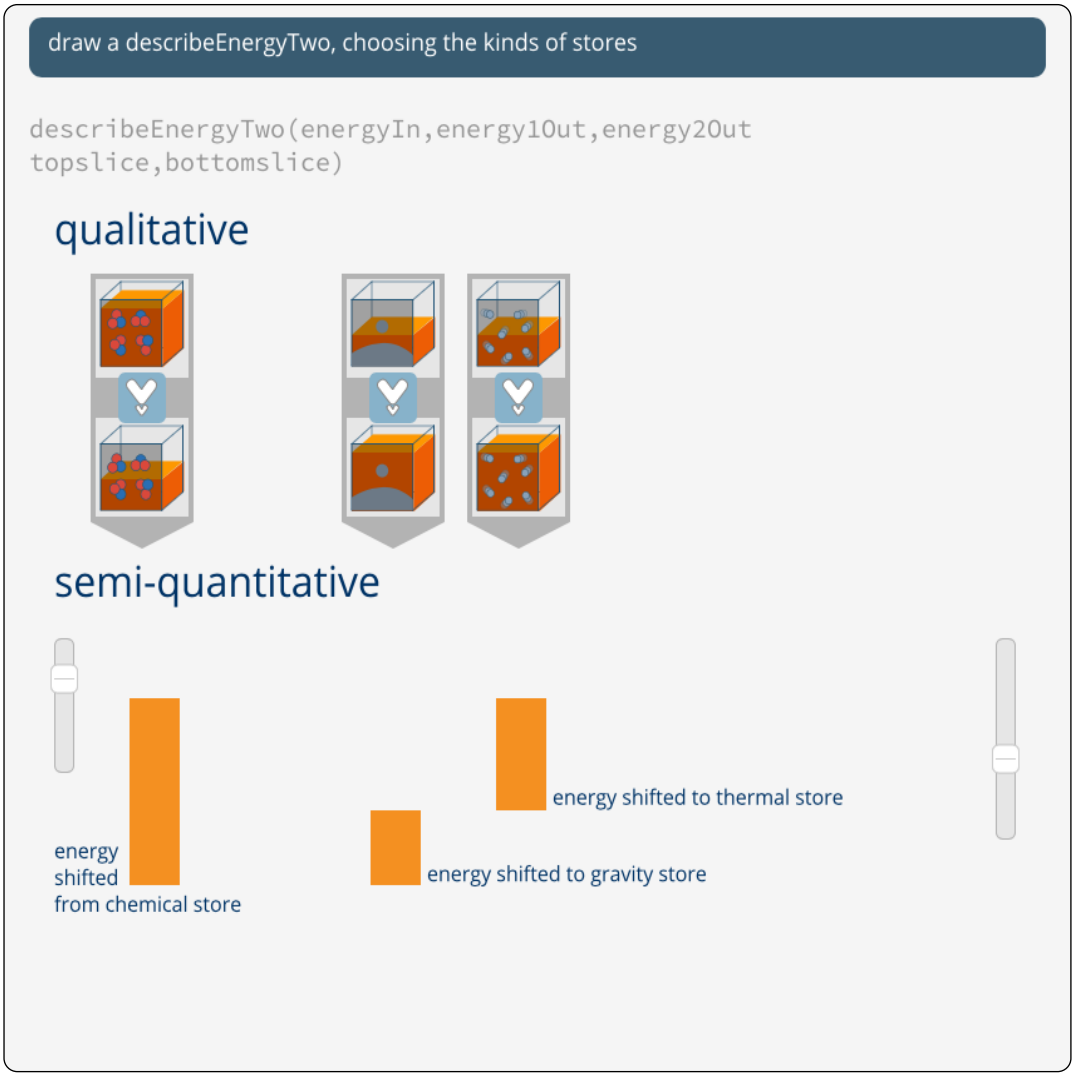
description

Requires a lot of supporting code- cf examples of use.

parameters

As above: sliders are often used to set the slices and input, and these sliders must be coded separately.

examples



describeEnergyThree

primitive and parameters

describeEnergyThree(energyIn,energy1Out,energy2Out,energy3Out,topslice,midslice,bottomslice)

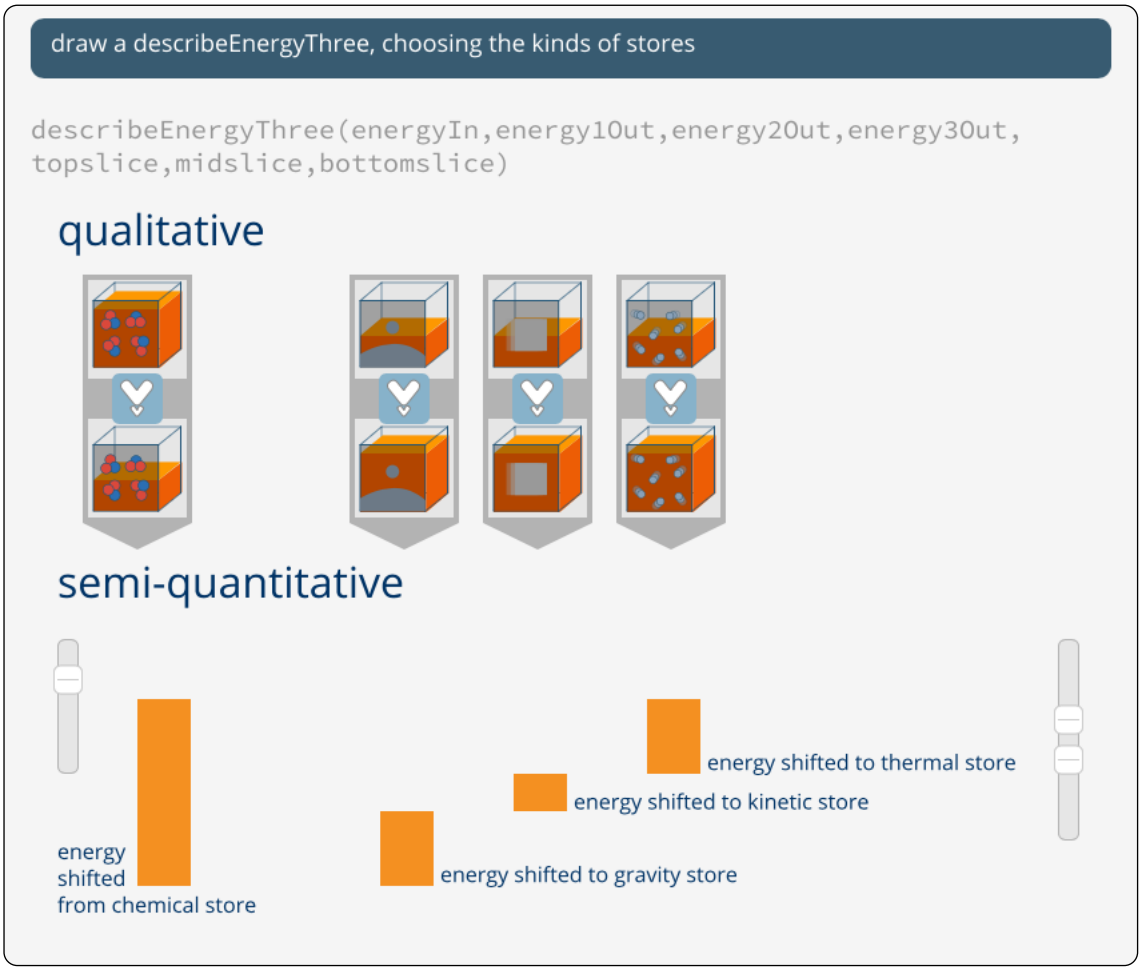
description

Requires a lot of supporting code- cf examples of use.

parameters

As above: sliders are often used to set the slices and input, and these sliders must be coded separately.

examples



Graphs and charts

pspace1D

primitive and parameters

`pspace1D(plotlabel,pheight,angle,valueNow)`

description

Draw a possibility space for one dimension, in which to show values of variables.

parameters

`plotlabel`: label for plot

`pheight`: the height of the area

`angle`: the angle for the plot (0 or 90)

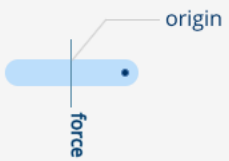
`valueNow`: current value to plot

examples

draw an pspace1D, of chosen quality, width and height

```
pspace1D(plotlabel,pheight,angle,valueNow)


pspace1D("force",50,90,20);
```



pspace2D

primitive and parameters

`pspace2D(p2Dheight,p2Dwidth,plotlabelX,plotlabelY,valueNowX,valueNowY)`

description

Draw a possibility space for two dimensions, in which to show values of variables.

parameters

`p2Dheight`: the height of the area

`p2Dwidth`: the width of the area

`plotlabelX`: label for plot

`plotlabelY`: label for plot

`valueNowX`: current value to plot

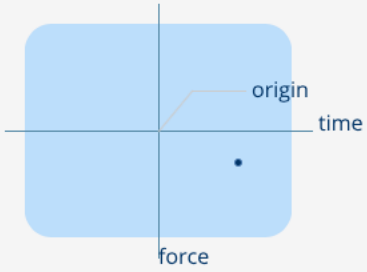
`valueNowY`: current value to plot

examples

draw an pspace2D, with chosen labels, width and height

```
pspace2D(p2Dheight,p2Dwidth,plotlabelX,plotlabelY,
valueNowX,valueNowY)
```

```
pspace2D(80,100,"force","time",50,20);
```



aspace

primitive and parameters

`aspace(aheight,awidth,plottitle,plotlabelX,plotlabelY,index-power,numberofsteps,stepsize)`

description

To demonstrate accumulations, graphically.

parameters

`aheight`: the height of the area

`awidth`: the width of the area

`plottitle`: label for plot

`plotlabelX`: label for plot

`plotlabelY`: label for plot

`indexpower`: power for the accumulation

`numberofsteps`: number of steps in the accumulation

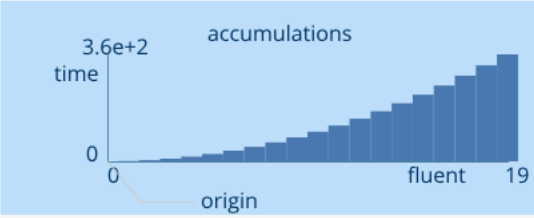
`stepsize`: size of steps in the accumulation

examples

draw an aspace, with chosen values

```
aspace(aheight,awidth,plottitle, plotlabelX,plotlabelY,
indexpower,numberofsteps,stepsize)

aspace(80, 100, "accumulations", "fluent", "time", 2, 20, 3);
```



fspace

primitive and parameters

`fspace(fwidth,fheight,plotlabel,valueNow)`

description

Draws a space on which to represent the values of functions.

parameters

`fwidth`: the width of the area

`fheight`: the height of the area

`plotlabel`: label for plot

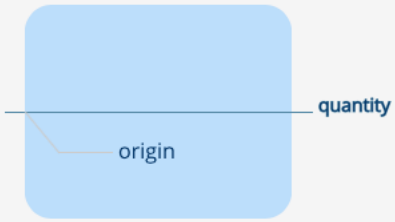
`valueNow`: current value to plot

examples

draw an fspace, with chosen values

```
fspace(fwidth,fheight,plotlabel,valueNow)

fspace(200, 80, "quantity", 12);
```



dspace

primitive and parameters

`dspace(dwidth,dheight,plottitle, plotlabelX,plotlabelY,indepVariable,depVariable)`

description

Draws a space on which to represent data values, help in the arrays indepVariable and depVariable. These need to be defined.

parameters

`dwidth`: the width of the area

`dheight`: the height of the area

`plottitle`: label for plot

`plotlabelX`: label for plot

`plotlabelY`: label for plot

`indepVariable`: name of array containing data

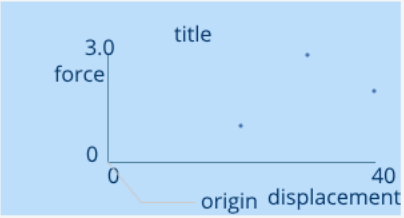
`depVariable`: name of array containing data

examples

draw an dspace, with chosen values

```
dspace(fwidth,fheight,plotlabel,valueNow)
```

```
dspace(200, 80, "title", "force", "displacement", 20, 12);
```



graphspace

primitive and parameters

`graphspace(fwidth,fheight,plotlabel)`

description

Draws a simple background, suitable for sparklines.

parameters

`fwidth`: the width of the area

`fheight`: the height of the area

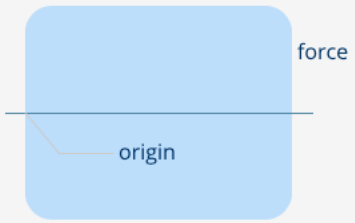
`plotlabel`: label for plot

examples

draw an graphspace, with chosen values

```
graphspace(fwidth,fheight,plotlabel)

graphspace(200, 80, "force");
```



SparkLineGraph

description

A class of objects to provide a spark line graph.

primitive and parameters

`XXX.create(xloc,yloc,slwidth,slheight,slLabel,plotKind,maxFixedVal-ue)`

parameters

`xloc,yloc`: give the location of the exploration puck, which explores the independent variable

`slwidth,slheight`: gives the dimensions of the graph.

`slLabel`: are the plot label

`plotKind`: can be one of "PN" for positive or negative values, autoscaling; "PO" for positive values, autoscaling; "FX" for fixed scale, when the value of maxFixedValue is used: for PO and PN this is not used

`maxFixedValue`: for fixed value graphs

preparation

This graph assumes that there is a MasterTicker running in the background of the PID, and that values will be fed into the sparkline.

setup

create the exploreRelationshipGraph(s) and set their location, in setup().

eg:

```
velcySparker = new SparkLineGraph;
velcySparker.create(20,300,300,80,"velocity","FX",30);
```

acquire data and plot

plot the graph in draw(), after acquiring the value to be plotted

eg:

```
velcySparker.acquire(30);
velcySparker.plot();
```

create an instance of a SparkLineGraph

```
velcySparker = new SparkLineGraph;
velcySparker.create(20,300,300,80,"velocity","FX",30);
```

Controllers and inputs

controlPuck

description

A class of objects to provide a controller to set two-dimensional data elegantly. Methods for returning the values determine the location of the puck and return two values between -0.5 and +0.5.

preparation

create a global variable which is the name for the controller(s)

eg:

```
var controllers="";
```

setup

create the controller(s) and set their location, in setup().

eg:

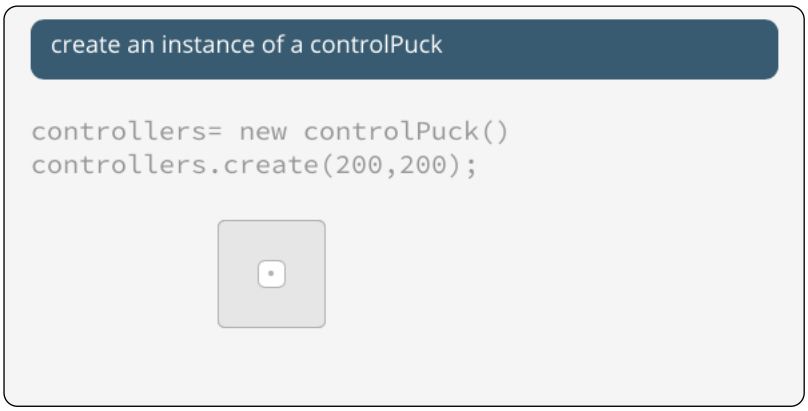
```
controllers= new controlPuck();
controllers.create(200,200);
```

get data

interrogate the controllers, in draw().

eg:

```
var displacements=createVector(controllers.getVal-
ues().xSet,controllers.getValues().ySet);
```



controlStripVerticalPositive

description

A class of objects to provide a controller to set one-dimensional, positive-only data elegantly. Methods for returning the values determine the location of the puck and return a value between 0 and 1.

preparation

create a global variable which is the name for the controller(s)

eg:

```
var controllers="";
```

setup

create the controller(s) and set their location, in setup().

eg:

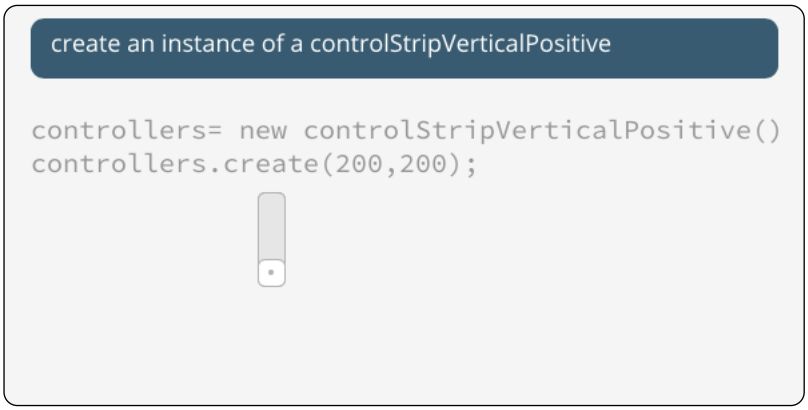
```
controllers= new controlStripVerticalPositive();
controllers.create(200,200);
```

get data

interrogate the controllers, in draw().

eg:

```
var displacements=createVector(controllers.getVal-
ues().xSet,controllers.getValues().ySet);
only the y value returns a varying value
```



controlStripHorizontalPositive

description

A class of objects to provide a controller to set one-dimensional, positive-only data elegantly. Methods for returning the values determine the location of the puck and return a value between 0 and 1.

preparation

create a global variable which is the name for the controller(s)

eg:

```
var controllers="";
```

setup

create the controller(s) and set their location, in setup().

eg:

```
controllers= new controlStripHorizontalPositive ();
controllers.create(200,200);
```

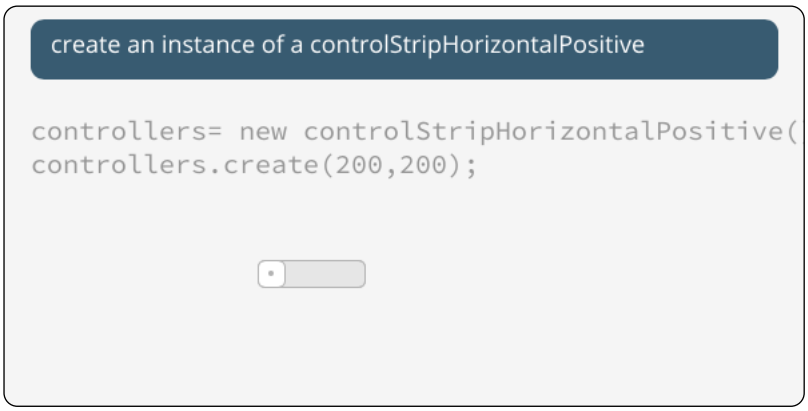
get data

interrogate the controllers, in draw().

eg:

```
var displacements=createVector(controllers.getVal-
ues().xSet,controllers.getValues().ySet);
```

only the x value returns a varying value



controlStripHorizontal

description

A class of objects to provide a controller to set one-dimensional data elegantly. Methods for returning the values determine the location of the puck and return a value between -0.5 and +0.5.

preparation

create a global variable which is the name for the controller(s)

eg:

```
var controllers="";
```

setup

create the controller(s) and set their location, in setup().

eg:

```
controllers= new controlStripHorizontal ();
controllers.create(200,200);
```

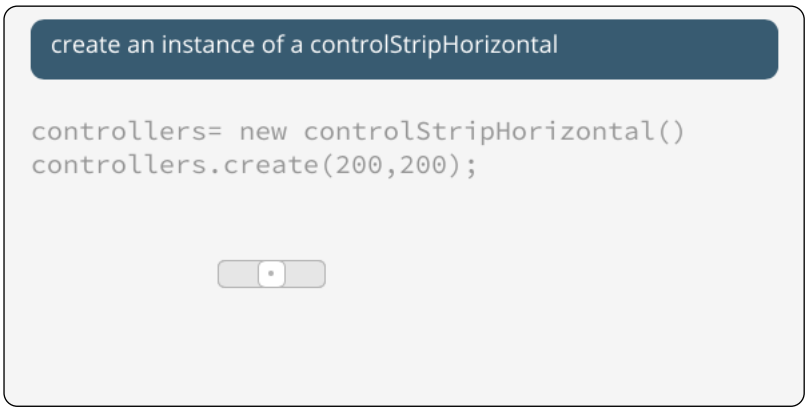
get data

interrogate the controllers, in draw().

eg:

```
var displacements=createVector(controllers.getVal-
ues().xSet,controllers.getValues().ySet);
```

only the x value returns a varying value



controlStripVertical

description

A class of objects to provide a controller to set one-dimensional data elegantly. Methods for returning the values determine the location of the puck and return a value between -0.5 and +0.5.

preparation

create a global variable which is the name for the controller(s)

eg:

```
var controllers="";
```

setup

create the controller(s) and set their location, in setup().

eg:

```
controllers= new controlStripVertical ();
controllers.create(200,200);
```

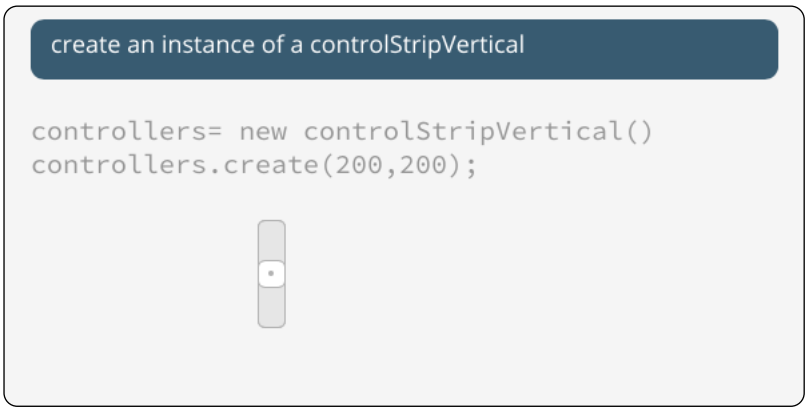
get data

interrogate the controllers, in draw().

eg:

```
var displacements=createVector(controllers.getVal-
ues().xSet,controllers.getValues().ySet);
```

only the y value returns a varying value



controlStripGraphIndependent

description

A class of objects to provide a 200 pixel long controller to set one-dimensional positive-only data elegantly, for the first quadrant of a graph. Methods for returning the values determine the location of the puck and return a value between 0 and 1.

preparation

create a global variable which is the name for the controller(s)

eg:

```
var controllers="";
```

setup

create the controller(s) and set their location, in setup().

eg:

```
controllers= new controlStripGraphIndependent ();
controllers.create(200,200);
```

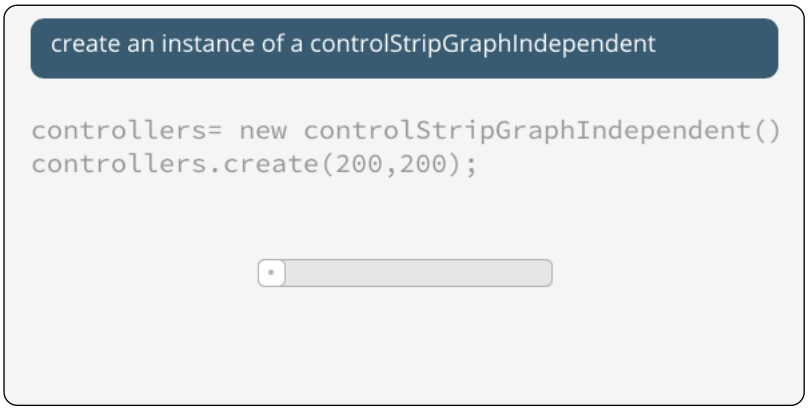
get data

interrogate the controllers, in draw().

eg:

```
var displacements=createVector(controllers.getVal-
ues().xSet,controllers.getValues().ySet);
```

only the x value returns a varying value



buttons(DOM)

These are DOM elements. Create in setup(), with a name which is their handle

simple buttons

```
mybutton = createButton("filling");
mybutton.position(50, 100);
mybutton.mousePressed(showit);
mybutton.class("PDLbutton");
```

showit is the name of the function invoked when the button is pressed, which may set the name of the button like this (in draw, or the function invoked by button press):

```
mybutton.html('filling');
```

The position is absolute, from the top left. The class is a CSS class, used to styling the button.

checkboxes

Also created in setup.

```
resultantbutton = createCheckbox("show resultant",false);
resultantbutton.position(560, height-50);
resultantbutton.class("PDLbutton");
```

test for the status in draw() like this:

```
if (resultantbutton.checked()){
    //do some stuff
}
```

controlButton

description

A class of objects to provide a clickable button.

preparation

create a global variable which is the name for the button

eg:

```
var climbutton;
```

setup

create the button and set its location and dimensions in setup().

eg:

```
climbutton= new controlButton(200,200,120,40);
```

show button

in Draw()

```
climbutton.drawButton();
```

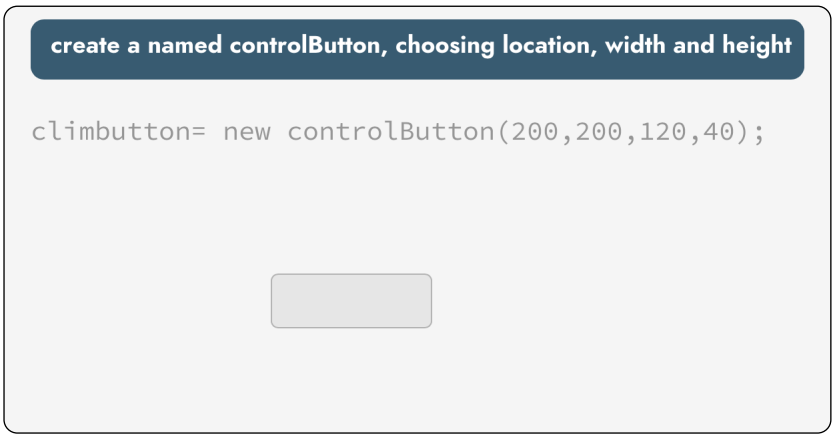
get data

add a listener function to change the state

```
function mouseReleased(){
    climbutton.checkPressed();
}
```

use this test to control the in

`climbutton.buttonwasPressed` will be either true or false, don't foget to reset this if you want to check for another button press.



labelButton

description

A class of objects to provide a clickable button with label.

preparation

create a global variable which is the name for the button

eg:

```
var labelledbutton;
```

setup

create the button and set its location and label in setup().

eg:

```
labelledbutton= new labelButton(200,200,"summer is here");
```

show button

in Draw()

```
labelledbutton.drawButton();
```

get data

add a listener function to change the state

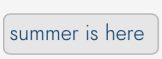
```
function mouseReleased(){
    labelledbutton.checkPressed();
}
```

use this test to control the in

`labelledbutton.buttonwasPressed` will be either true or false, don't foget to reset this if you want to check for another button press.

create a named labelButton, choosing location and label

```
labelledbutton= new labelButton(200,200,
"summer is here");
```



checkButton

description

A class of objects to provide a clickable checkbox button with label.

preparation

create a global variable which is the name for the button

eg:

```
var doitbutton;
```

setup

create the button and set its location,label and initial state(checked or not) in setup().

eg:

```
doitbutton = new checkButton(200,200,"sum of all fears",true);
```

show button

in Draw()

```
doitbutton.drawButton();
```

get data

add a listener function to change the state

```
function mouseReleased(){
    doitbutton.changeState();
}
```

use this test to control the in

`doitbutton.buttonisChecked` will be either true or false

sliders

description

A class of objects customised one-dimensional controllers, separate from the DOM sliders, which are not used. They may have one or more handles and can therefore be used as dividers. They return a value between 0 and 1. You can set the number of possible steps along the slider.

primitive and parameters

`IanSlider(xPos,yPos,sliderheight,sliderwidth,noOfSteps,handle-postions,showlabel)`

parameters

`xPos`: the location of the slider

`yPos`: the location of the slider

`slider height`: the height of the slider

`slider width`: the width of the slider

`no of steps`: restrict the slider to a number of steps between the maximum and minimum locations

`handle points`: an array to set the initial locations of the handles, eg[.3,.7]

`show label`: either true or false to show the values

setup

create the controller(s) and set their location, in setup(), with a name which is the handle for the slider.

eg:

```
slider=new IanSlider(400,100,200,20,0,[0.2,0.5],true);
```

create two functions to handle mouseclicks on the slider:

```
function mousePressed(){
        slider.mousePressed();
    }
function mouseReleased(){
        slider.mouseReleased();
    }
```

set handle positions or get data

slider.setValue(0.2): sets value for handle with index 0

slider.setValue(1,0.5): sets value for handle with index 1.

slider.getValue(): gets value of handle with index 0.May write as slider.getValue(0);

slider.getValue(1): gets value of handle with index 1.

Graphs with controllers

graphExploreBoxArea

description

A class of objects to provide a graph and controller to set to explore areas on the graph.

primitive and parameters

```
XXX.create(xloc,yloc,graphHeight,graphWidth,plotlabelX,plotlabelY)
```

parameters

`xloc,yloc`: give the location of the exploration puck

`graphSize`: gives the dimensions of the graph (200 works well, the offsets burnt into the class may need adjusting if you use very different values.

`quadrants`: can be 1,2,or 4, to set the number of quadrants that can be explored.

`plotlabelX,plotlabelY`: are the axis labels and can include pml

setup

create the graphExploreBoxArea(s) and set their location, in setup().

eg:

```
exploreThisGraph= new graphExploreBoxArea;
exploreThisGraph.create(60,230,200,4,"force / N","velocity / JustU-
nit{m s -1}");
exploreThisGraph.setLabels();
```

explore graph and get data

explore the graph in draw().

eg:

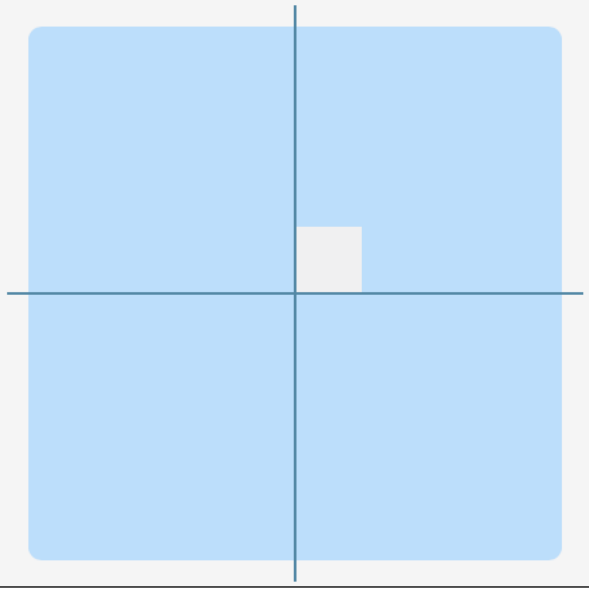
```
exploreThisGraph.draw();
```

And maybe, to get the values out, to show representations:

```
exploreThisGraph.getValues().boxHeight
exploreThisGraph.getValues().boxWidth
```

create an instance of a graphExploreBoxArea

```
exploreThisGraph= new graphExploreBoxArea;
exploreThisGraph.create(90,100,200,4,"force / N",
"velocity / JustUnit{m s -1}");
exploreThisGraph.setLabels();
```



exploreRelationshipGraph

description

A class of objects to provide a graph and controller to set to explore relationships on a graph.

primitive and parameters

```
XXX.create(xloc,yloc,graphSize,quadrants,plotlabelX,plotlabelY)
```

parameters

`xloc,yloc`: give the location of the exploration puck, which explores the independent variable

`graphSize`: gives the dimensions of the graph (200 works well, the offsets burnt into the class may need adjusting if you use very different values.

`quadrants`: can be 1,2,3or 4, to set the quadrants that can be explored.

`plotlabelX,plotlabelY`: are the axis labels and can include pml

setup

create the exploreRelationshipGraph(s) and set their location, in setup().

eg:

```
exploreThisGraph= new exploreRelationshipGraph;
exploreThisGraph.create(60,350,200,3,"distortion / m","tension / N");
exploreThisGraph.setLabels();
```

explore graph and get data

explore the graph in draw(), after setting the relationship to be explored

eg:

```
exploreThisGraph.draw();
    var k = 7;
    var x = -exploreThisGraph.getValues().xSet*20;
    var F = k*x;
    exploreThisGraph.plotValues(x,F);
```

create an instance of a exploreRelationshipGraph

```
exploreThisGraph= new exploreRelationshipGraph;
exploreThisGraph.create(60,350,200,3,
"distortion / m","tension / N");
exploreThisGraph.setLabels();
```

Accumulation engines

predictAccumulate()

predictAccumulate()

rk4

rk4(currentposition, currentspeed, calculatedacceleration)

People adornments

joeHead

primitive and parameters

`joeHead()`

description

The head of Joe, a guide.

parameters

none

examples

draw the head of Joe

```
joeHead()
```

```
joeHead()
```

melHead

primitive and parameters

`melHead()`

description

The head of Mel, a guide.

parameters

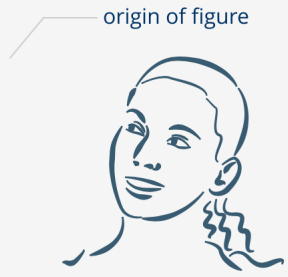
none

examples

```
draw the head of Mel

melHead()


melHead()
```



sueHead

primitive and parameters

`sueHead()`

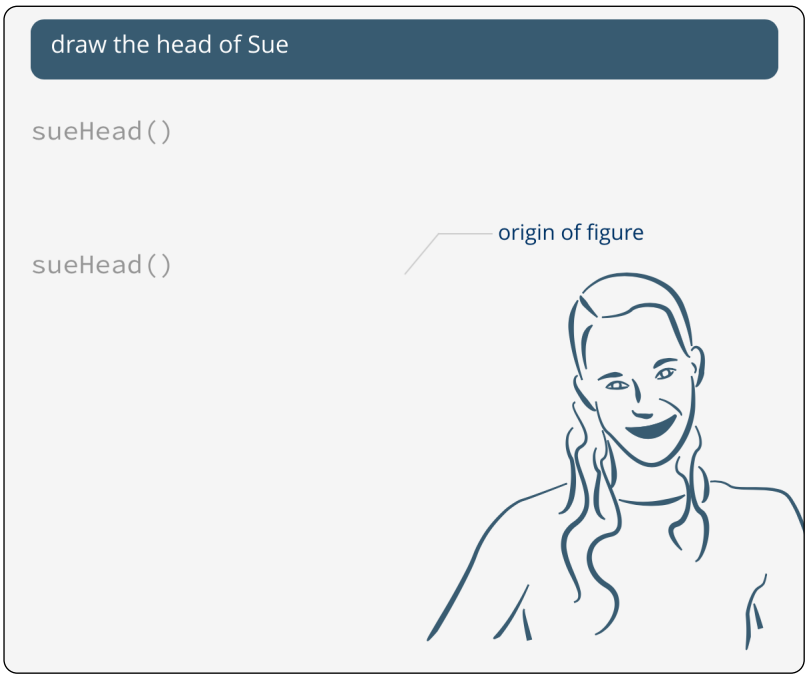
description

The head of Sue, a guide.

parameters

none

examples



drawGirl1

primitive and parameters

`drawGirl1()`

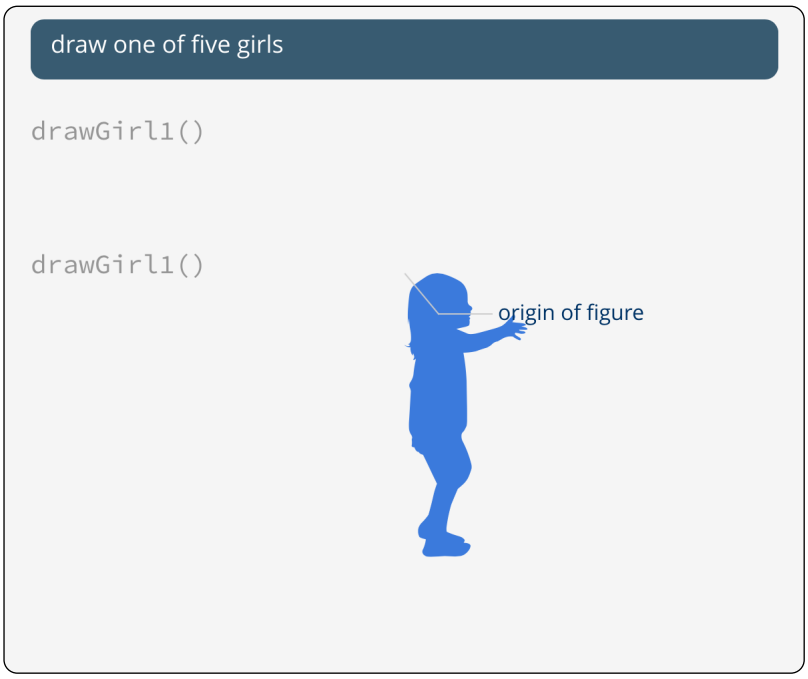
description

Draws an outline of a girl.

parameters

none

examples



drawGirl2

primitive and parameters

`drawGirl2()`

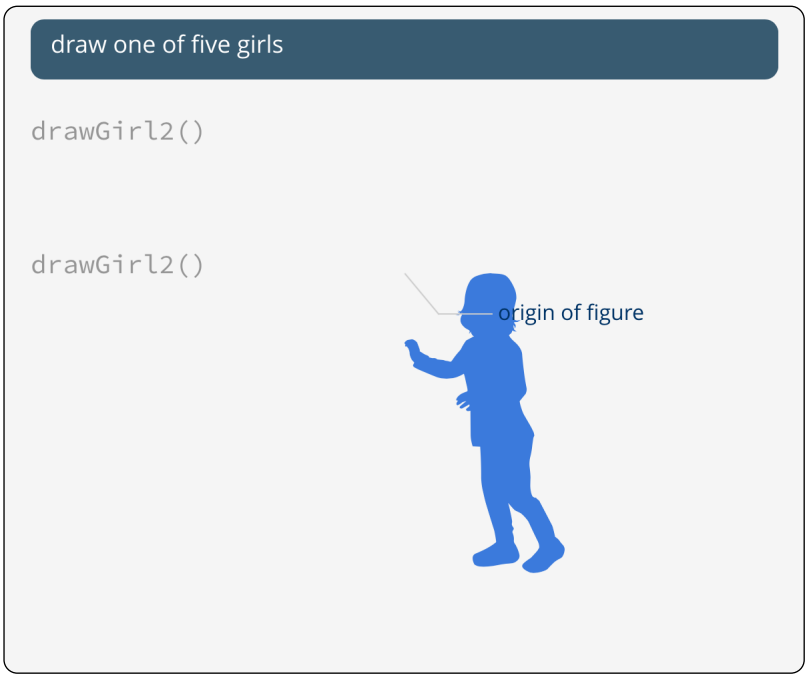
description

Draws an outline of a girl.

parameters

none

examples



drawGirl3

primitive and parameters

`drawGirl3()`

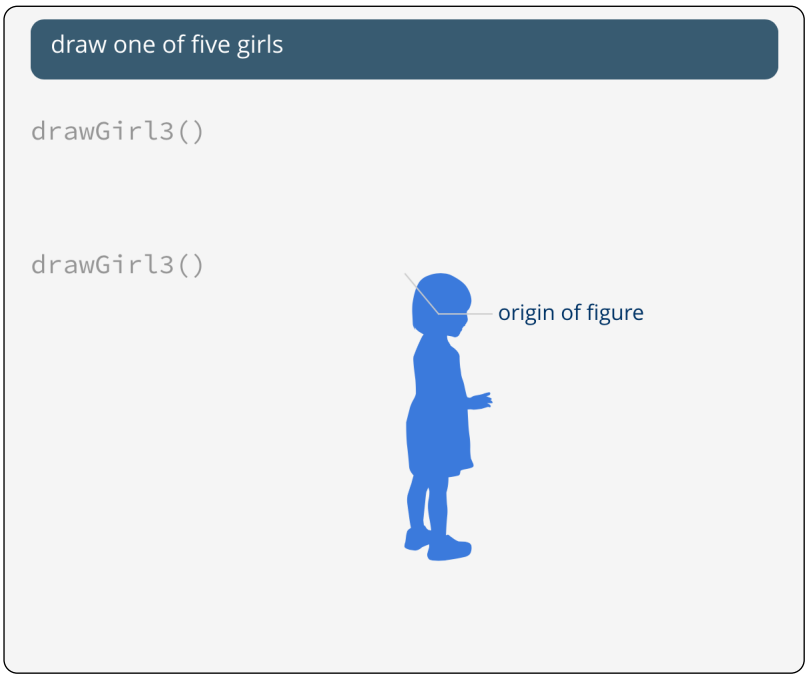
description

Draws an outline of a girl.

parameters

none

examples



drawGirl4

primitive and parameters

`drawGirl4()`

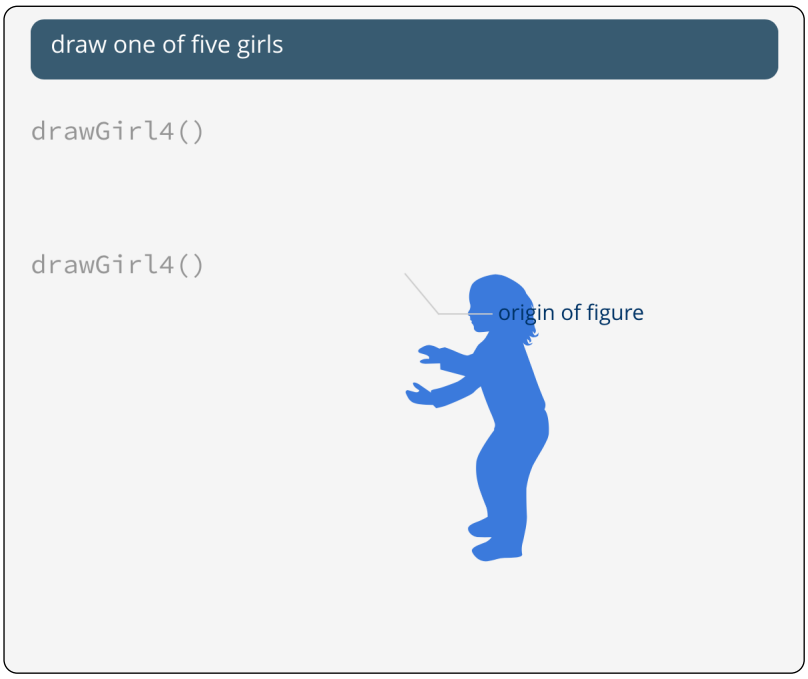
description

Draws an outline of a girl.

parameters

none

examples



drawGirl5

primitive and parameters

`drawGirl5()`

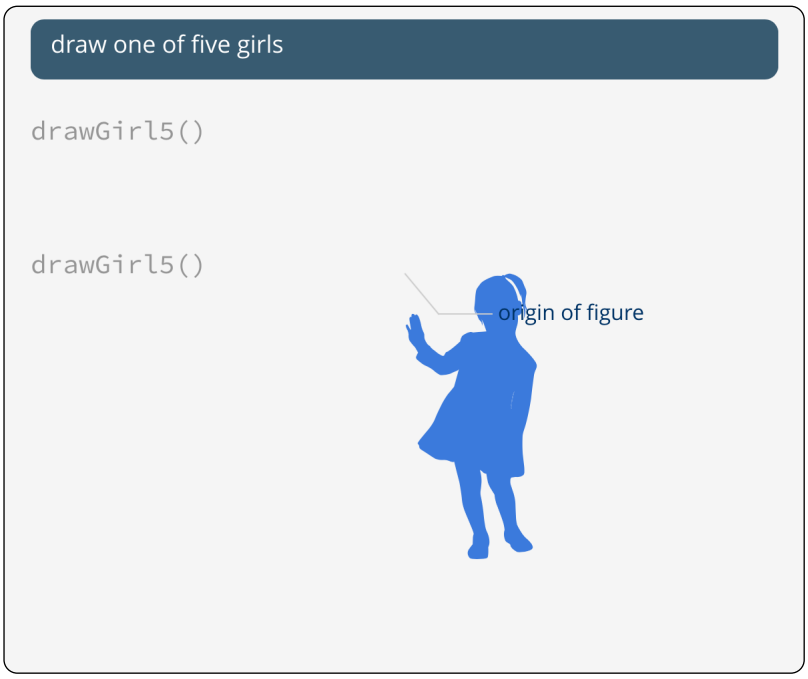
description

Draws an outline of a girl.

parameters

none

examples



drawBoy1

primitive and parameters

`drawBoy1()`

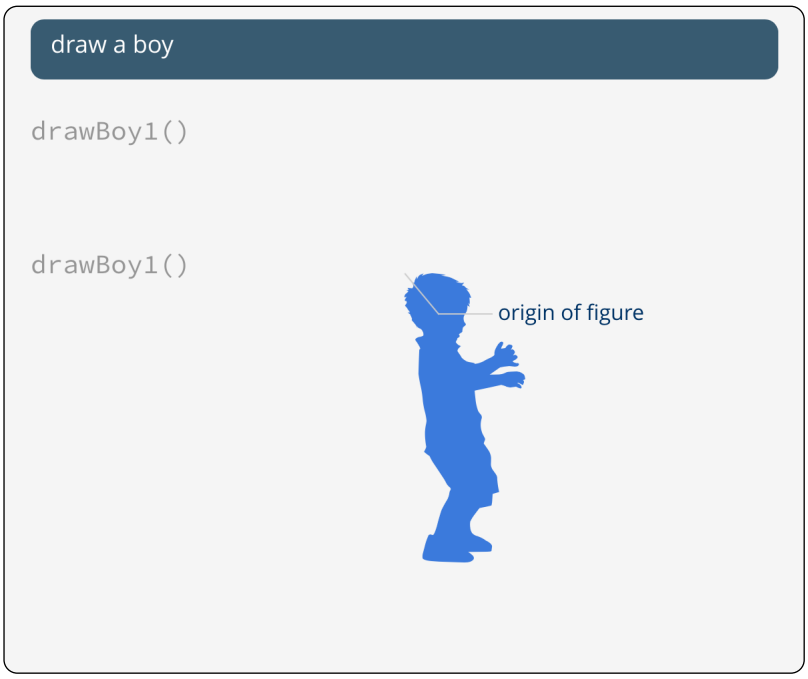
description

Draws an outline of a boy.

parameters

none

examples



drawCar1

primitive and parameters

`drawCar1()`

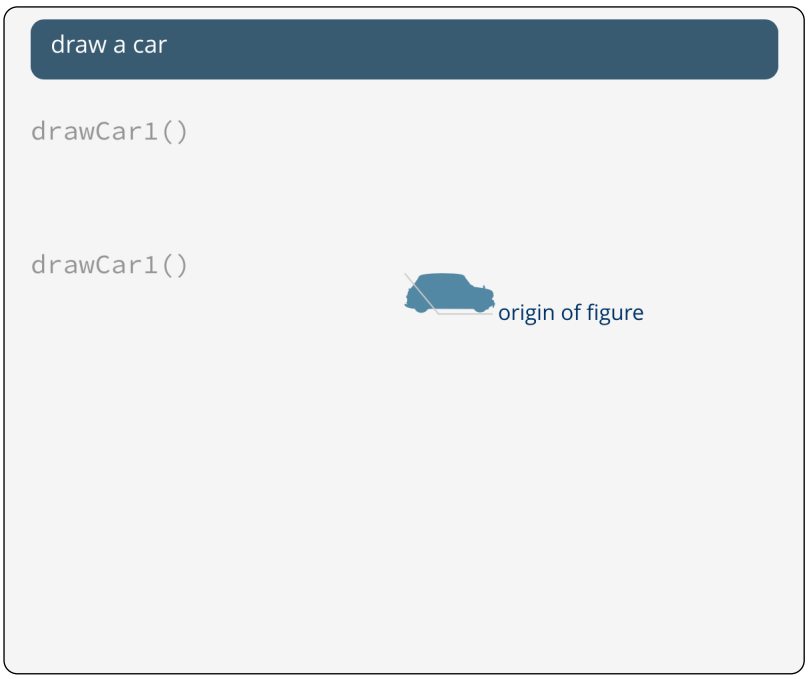
description

Draws an outline of a car.

parameters

none

examples



drawFingerArm1

primitive and parameters

`drawFingerArm1()`

description

Draws an outline of a stylised pointing arm and finger.

parameters

none

examples

draw pointing finger and arm

```
drawFingerArm1()


drawFingerArm1()
```

drawThumbsUp1

primitive and parameters

`drawThumbsUp1()`

description

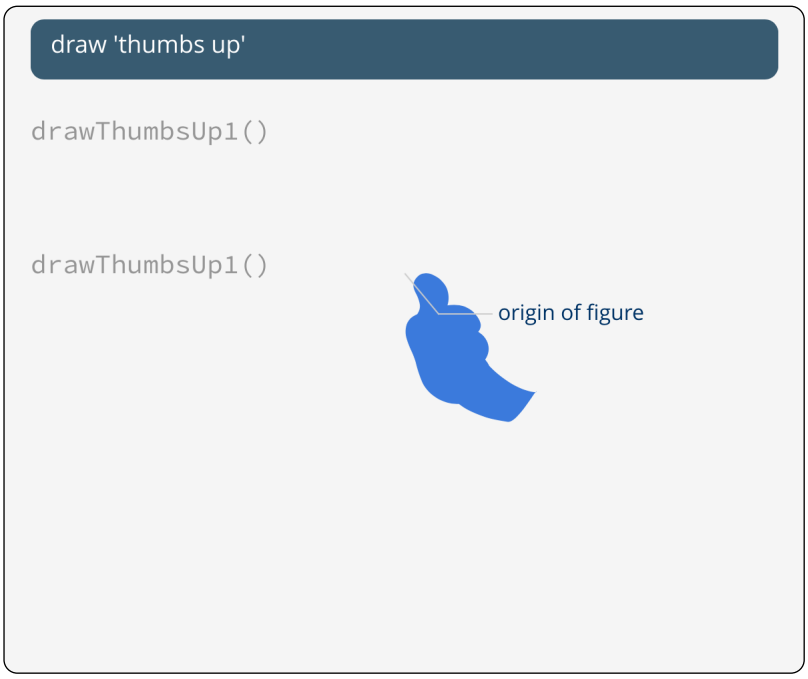
Draws an outline of a stylised thumbs up.

parameters

none

examples

```
```



drawTiFoThinking

primitive and parameters

`drawTiFoThinking()`

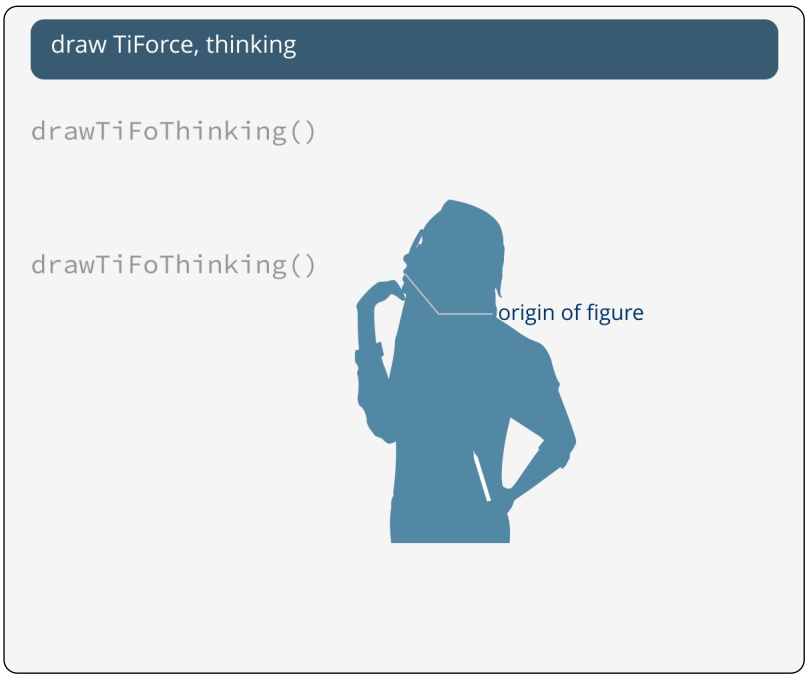
description

Tiffany, of the force tribe, does thinking.

parameters

none

examples



drawTiFoPointing

primitive and parameters

`drawTiFoPointing()`

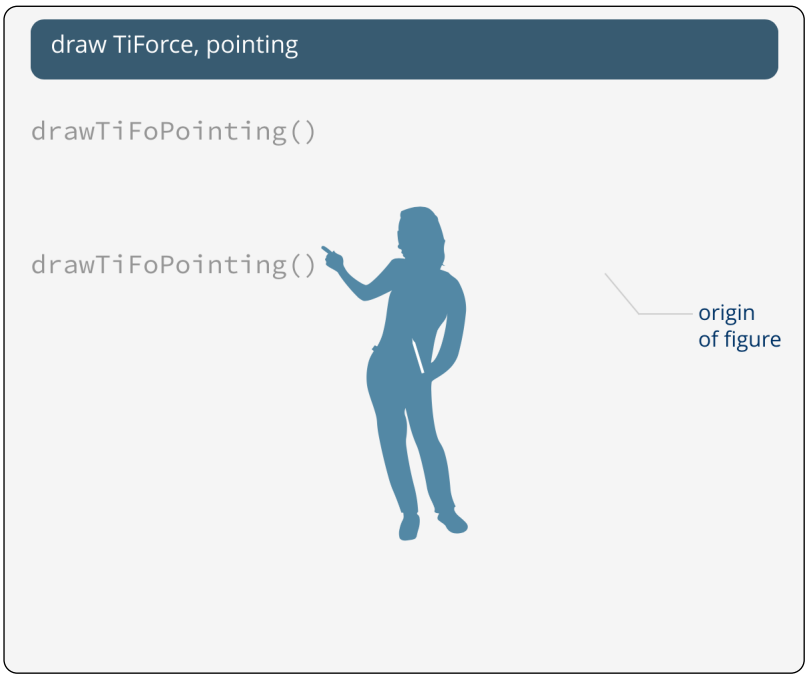
description

Tiffany, of the force tribe, does pointing.

parameters

none

examples



drawTiFoSpin

primitive and parameters

`drawTiFoSpin()`

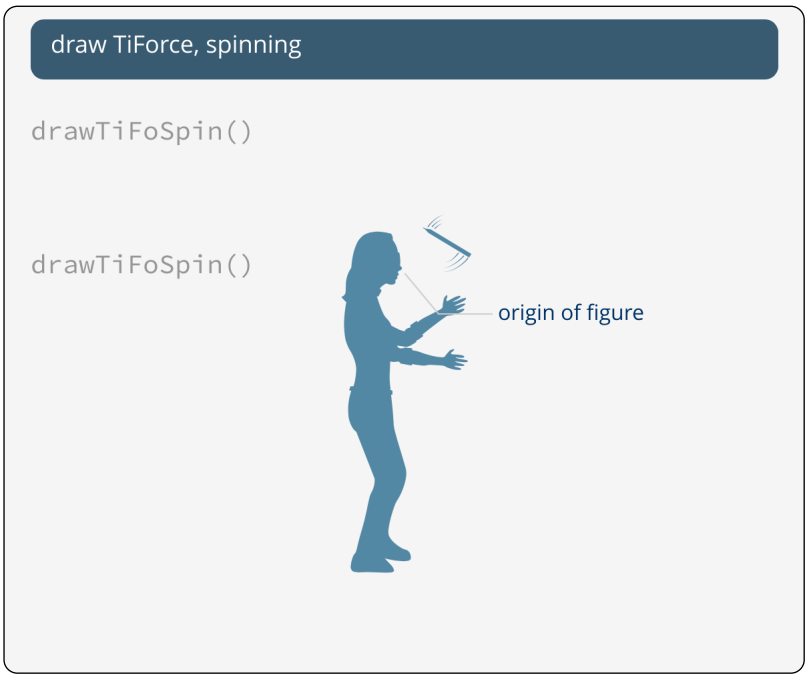
description

Tiffany, of the force tribe, does waiting.

parameters

none

examples



discoursedroid

primitive and parameters

`discoursedroid(personid,xloc,yloc,flip)`

description

parameters

`personid:` a number. between 0 and 11, to choose the droid

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`flip:` boolean to determine whether flipped left to right or not

examples

draw a discoursedroid, choosing type, location and orientation

```
discoursedroid(personid,xloc,yloc,flip)

discoursedroid(8,180,300,false)
```

structural adornments

physicalPane

primitive and parameters

`physicalPane(xloc, yloc, paneWidth, paneHeight )`

description

Draws a physical pane.

parameters

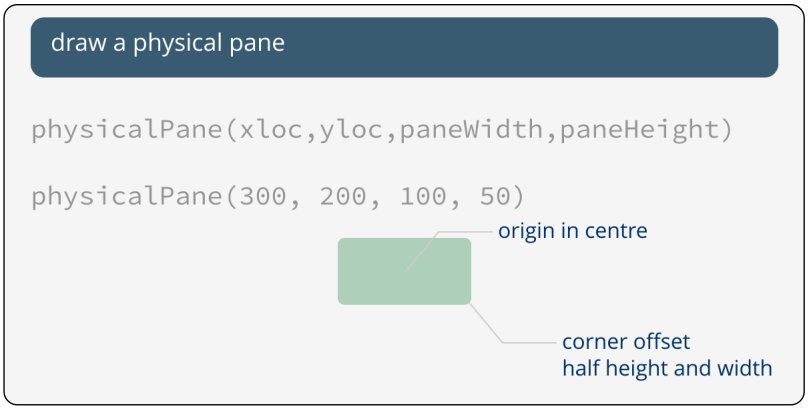
`xloc`

`yloc`

`paneWidth`

`paneHeight`

examples



conceptualPane

primitive and parameters

`conceptualPane(xloc, yloc, paneWidth, paneHeight )`

description

parameters

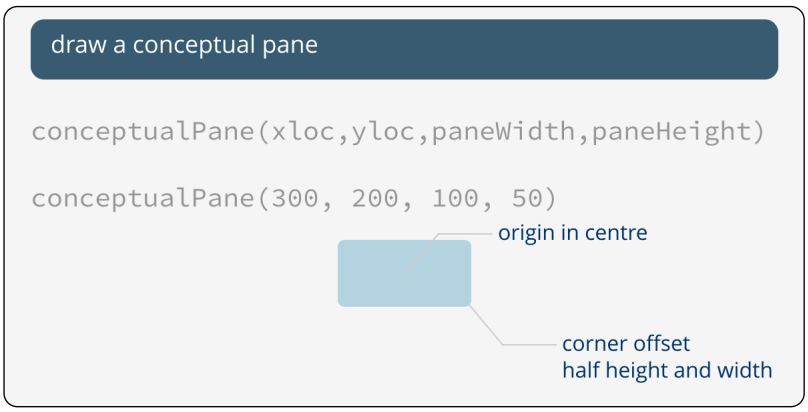
`xloc`

`yloc`

`paneWidth`

`paneHeight`

examples



paneBadgeSnapshot

primitive and parameters

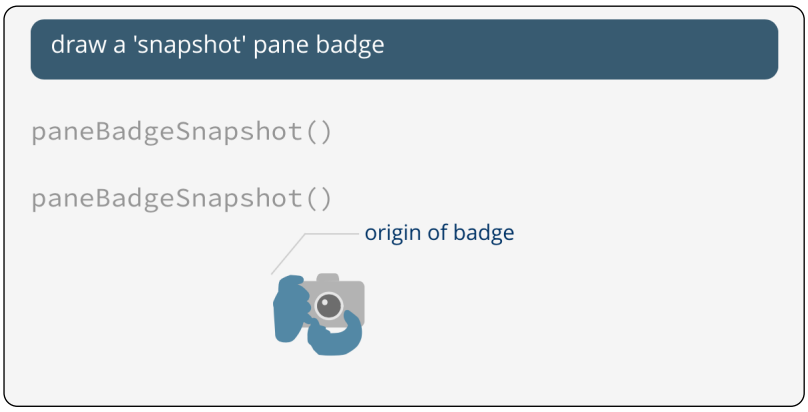
paneBadgeSnapshot()

description

parameters

none

examples



paneBadgeIntervene

primitive and parameters

`paneBadgeIntervene()`

description

parameters

none

examples

draw a 'intervene' pane badge

```
paneBadgeIntervene()
```

```
paneBadgeIntervene()
```

transitionEvolve

primitive and parameters

`transitionEvolve(xloc,yloc, rotation)`

description

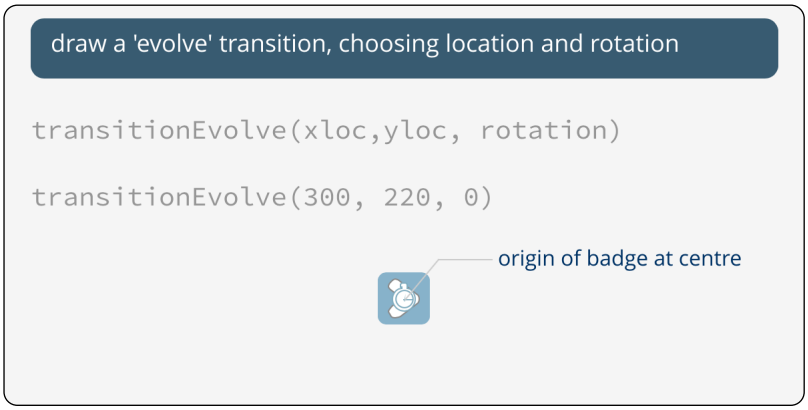
`xloc`

`yloc`

`rotation`

parameters

examples



transitionIntervene

primitive and parameters

`transitionIntervene(xloc, yloc,  rotation)`

description

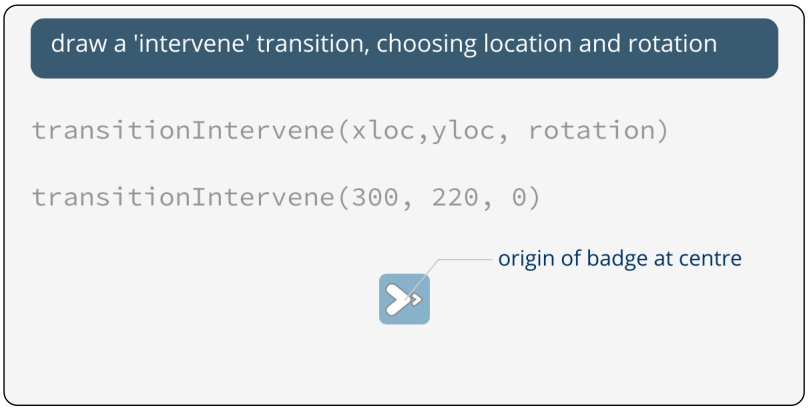
parameters

`xloc`

`yloc`

`rotation`

examples



transitionRedescribe

primitive and parameters

`transitionRedescribe(xloc,yloc, rotation)`

description

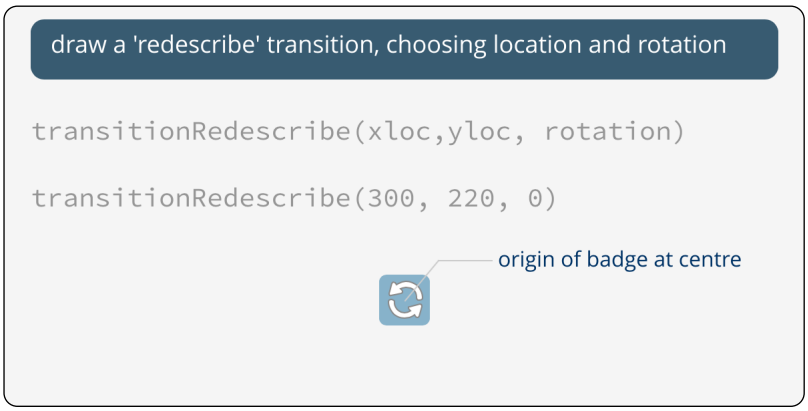
parameters

`xloc`

`yloc`

`rotation`

examples



transitionStep

primitive and parameters

`transitionStep(xloc,yloc, rotation)`

description

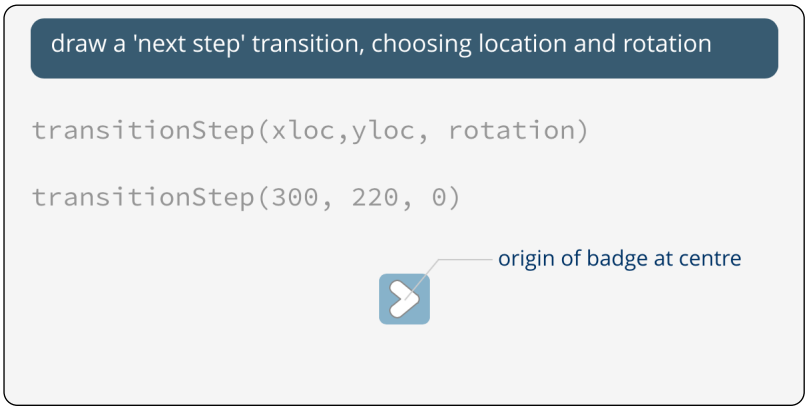
parameters

`xloc`

`yloc`

`rotation`

examples



SIDPane

primitive and parameters

`SIDPane(paneKind, paneNumber, paneAction, titleText)`

description

parameters

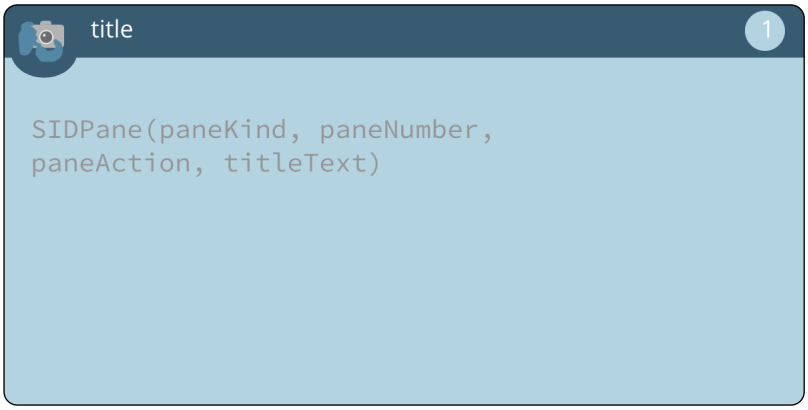
`paneKind:` one of "C" or "P" for conceptual or physical

`paneNumber:` a character string to incate the place of the pane in a sequence

`paneAction:` One of "S" or "F" to generate asnapshot or intervention pane badge

`titleText`: a string to set the title of the pane

examples



Named text blocks, styled by css

Isolated words on a line or relationships or a part of a diagram dealt with by specialised functions.

Asking these to draw each time slows the execution of the draw cycle down a lot, so place them in set-up() if not a static diagram.

justWords

primitive and parameters

`justWords(textSet,classSet,xloc,yloc,widthSet)`

description

Puts a CSS-styled text bloc in the DOM, with randomly assigned name (hopefully unique). CSS styles are: infoBlock,posterinfoBlock,passComment,passCommentL,passCommentR,posterpassComment, posterpassCommentL,posterpassCommentR,posterTitle,thinkingR,thinkingL,chattercommentR,chattercommentL,chatterthinkingR,chatterthinkingL,topchatterthinkingL,topchatterthinkingR,topchattercommentR,topchattercommentL.

parameters

`textSet`: the words

`classSetkind`: the css style

`xloc`: where

`yloc`: where

`setWidth`: the width of the block

examples

draw some justWords

```
justWords(textSet,classSet,xloc,yloc,widthSet)

justWords("say something", "passCommentL",
20, 200, 300)
```

headWords

primitive and parameters

`headWords(kind,WhoSpeakspeakThis,xloc,yloc,setWidth,sideHead)`

description

Gives you one of Joe, Sue or Mel as to 'talk' or 'think' either 'L' or 'R' with a CSS-styled text block in the DOM, with randomly assigned name (hopefully unique).

CSS styles are: infoBlock,posterinfoBlock,passComment,passCommentL,passCommentR,posterpassComment, posterpassCommentL,posterpassCommentR,posterTitle,thinkingR,thinkingL,chattercommentR,chattercommentL,chatterthinkingR,chatterthinkingL,topchatterthinkingL,topchatterthinkingR,topchattercommentR,topchattercommentL.

parameters

`kind`: the css style

`WhoSpeak`: 'Joe', 'Sue' or 'Mel'

`speakThis`: words to say

`xloc`: where

`yloc`: where

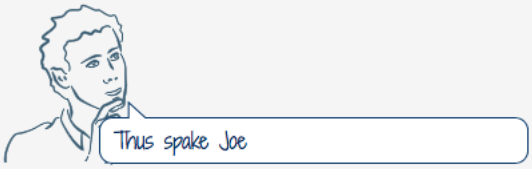
`setWidth`: the width of the block

`sideHead`: 'L' or 'R'

examples

draw some headWords

```
headWords(kind,WhoSpeakspeakThis,xloc,yloc,
setWidth,sideHead)
justWords("say something", "passCommentL",
20, 200, 300)
```



words in the diagrams

speechbubble

primitive and parameters

`speechbubble(speakThis,xloc,yloc,tailLoc)`

description

Places a flexible speech bubble containing the words at the location specified by locx and locy, with a tail set by "YY". "YY" can be one of "TR", "TL", "BR", "BL". You can force a line split by inserting \n.

parameters

`speakThis:` a string to set what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

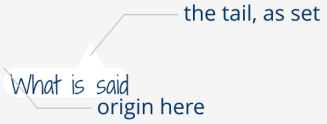
`tailLoc`: one of "TR", "TL", "BR", "BL"

examples

draw a speech bubble, choosing location and tail location

```
speechbubble(speakThis,xloc,yloc,tailLoc)

speechbubble("What is said.",40,230,"TL")
```



talkbubbleBL

primitive and parameters

`talkbubbleBL(sayThis,xloc,yloc, bubblewidth,bubbleheight)`

description

Places a speech bubble containing the words at the location specified by locx and locy, with a tail such that the bubble is below the origin and to the left. You can force a line split by inserting \n. pml is not available in the text, only straight text.

parameters

`sayThis:` a string to set what is said

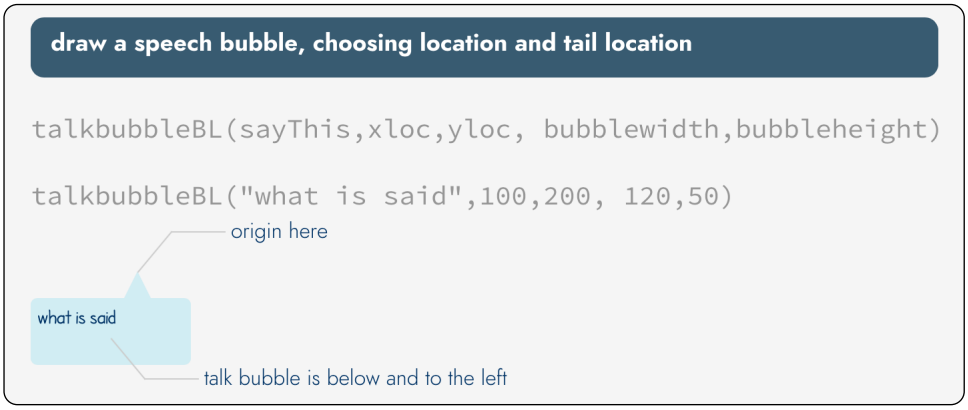
`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: width of bubble in pixel

`bubbleheight`: width of bubble in pixel

examples



talkbubbleBR

primitive and parameters

`talkbubbleBR(sayThis,xloc,yloc, bubblewidth,bubbleheight)`

description

Places a speech bubble containing the words at the location specified by locx and locy, with a tail such that the bubble is below the origin and to the right. You can force a line split by inserting \n. pml is not available in the text, only straight text.

parameters

`sayThis:` a string to set what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: width of bubble in pixel

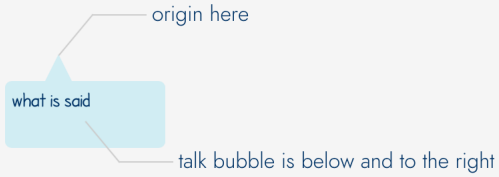
`bubbleheight`: width of bubble in pixel

examples

draw a speech bubble, choosing location and tail location

```
talkbubbleBR(sayThis,xloc,yloc, bubblewidth,bubbleheight)

talkbubbleBR("what is said",100,200, 120,50)
```



talkbubbleTL

primitive and parameters

`talkbubbleTL(sayThis,xloc,yloc, bubblewidth,bubbleheight)`

description

Places a speech bubble containing the words at the location specified by locx and locy, with a tail such that the bubble is above the origin and to the left. You can force a line split by inserting \n. pml is not available in the text, only straight text.

parameters

`sayThis:` a string to set what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: width of bubble in pixel

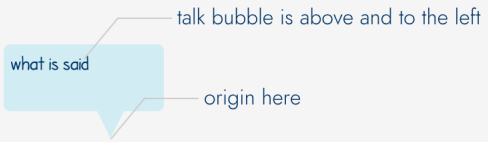
`bubbleheight`: width of bubble in pixel

examples

draw a speech bubble, choosing location and tail location

```
talkbubbleTL(sayThis,xloc,yloc, bubblewidth,bubbleheight)

talkbubbleTL("what is said",100,200, 120,50)
```



talkbubbleTR

primitive and parameters

`talkbubbleTR(sayThis,xloc,yloc, bubblewidth,bubbleheight)`

description

Places a speech bubble containing the words at the location specified by locx and locy, with a tail such that the bubble is above the origin and to the right. You can force a line split by inserting \n. pml is not available in the text, only straight text.

parameters

`sayThis:` a string to set what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: width of bubble in pixel

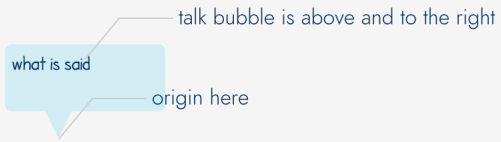
`bubbleheight`: width of bubble in pixel

examples

draw a speech bubble, choosing location and tail location

```
talkbubbleTR(sayThis,xloc,yloc, bubblewidth,bubbleheight)

talkbubbleTR("what is said",100,200, 120,50)
```



thoughtbubbleTL

primitive and parameters

`thoughtbubbleTL(thinkThis,xloc,yloc, bubblewidth,bubbleheight)`

description

Places a speech bubble containing the words at the location specified by locx and locy, with a tail such that the bubble is above the origin and to the left. You can force a line split by inserting \n. pml is not available in the text, only straight text.

parameters

`thinkThis:` a string to set what is thought

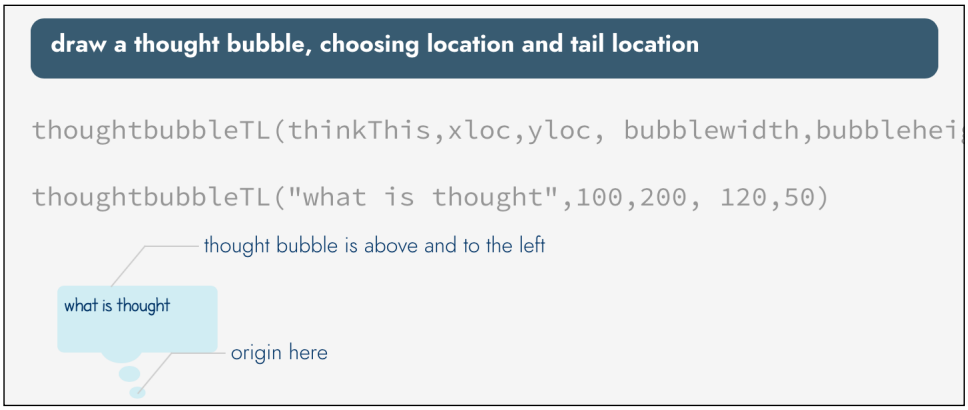
`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: width of bubble in pixel

`bubbleheight`: width of bubble in pixel

examples



thoughtbubbleTR

primitive and parameters

`thoughtbubbleTR(thinkThis,xloc,yloc, bubblewidth,bubbleheight)`

description

Places a speech bubble containing the words at the location specified by locx and locy, with a tail such that the bubble is above the origin and to the right. You can force a line split by inserting \n. pml is not available in the text, only straight text.

parameters

`thinkThis:` a string to set what is thought

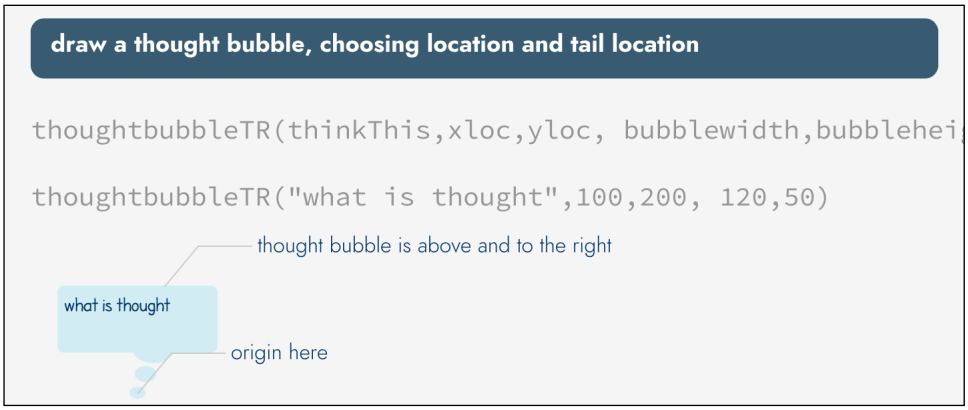
`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: width of bubble in pixel

`bubbleheight`: width of bubble in pixel

examples



DIPFheadL

primitive and parameters

`DIPFheadL(speakThis,xloc,yloc,bubblewidth)`

description

Draw a speaking head, choosing location,bubble width, tail location.

parameters

`speakThis`: what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`bubblewidth`: the width of the speech bubble

examples

draw a speaking head, with choices

```
DIPFheadL(speakThis,xloc,yloc,bubblewidth)

DIPFheadL("say something", 200,180, 100)
```

DIPMheadL

primitive and parameters

`DIPMheadL(speakThis,xloc,yloc,bubblewidth)`

description

parameters

`speakThis`: what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

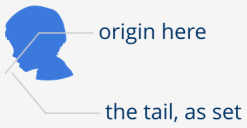
`bubblewidth`: the width of the speech bubble

examples

draw a speaking head, with choices

```
DIPMheadL(speakThis,xloc,yloc,bubblewidth)

DIPMheadL("say something", 200,180, 100)
```



DIPFheadR

primitive and parameters

`DIPFheadR(speakThis, xloc, yloc,bubblewidth)`

description

parameters

`speakThis`: what is said

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

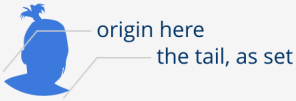
`bubblewidth`: the width of the speech bubble

examples

draw a speaking head, with choices

```
DIPFheadR(speakThis,xloc,yloc,bubblewidth)

DIPFheadR("say something", 200,180, 100)
```



saydroid

primitive and parameters

`saydroid(personid,xloc,yloc,speakThis,bubblewidth,bubble-height,bubbleloc)`

description

parameters

`personid:` a number. between 0 and 11, to choose the droid

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`speakThis:` a string to set what is said

`bubblewidth`: width of bubble in pixel

`bubbleheight`: width of bubble in pixel

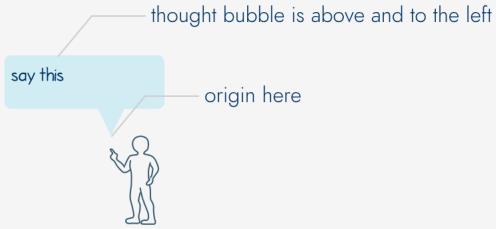
`bubbleloc`: location of bubble, one of "L" or "R"

examples

draw a speak droid, choosing type, location and orientation

```
saydroid(personid,xloc,yloc,speakThis,bubblewidth,
bubbleheight,bubbleloc)

saydroid(1,180,300,"say this",120,40,"L")
```



thinkdroid

primitive and parameters

`thinkdroid(personid,xloc,yloc,thinkThis,bubblewidth,bubble-height,bubbleloc)`

description

parameters

`personid:` a number. between 0 and 11, to choose the droid

`xloc:` where the x-origin is placed

`yloc:` where the y-origin is placed

`thinkThis:` a string to set what is said

`bubblewidth`: width of bubble in pixel

`bubbleheight`: width of bubble in pixel

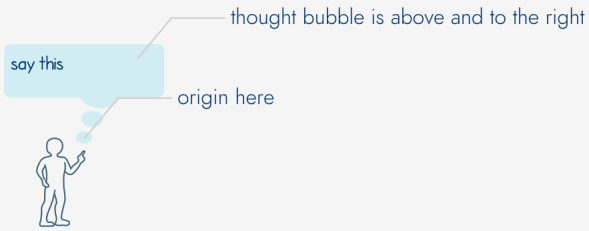
`bubbleloc`: location of bubble, one of "L" or "R"

examples

draw a thought droid, choosing type, location and orientation

```
thinkdroid(personid,xloc,yloc,thinkThis,bubblewidth,
bubbleheight,bubbleloc)

thinkdroid(5,180,300,"think this",120,40,"R")
```



words

primitive and parameters

`words(thisLabel,xloc,yloc)`

description

parameters

`thisLabel`: the words

`xloc:` where the x-origin is placed

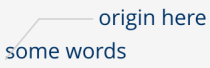
`yloc:` where the y-origin is placed

examples

write words, with choices

```
words(thisLabel,xloc,yloc)

words("some words", 200,230)
```



wordSuper

primitive and parameters

`wordSuper(thisLabel,thisSuperScript,xloc,yloc)`

description

parameters

`thisLabel`: the words

`thisSuperScript`: the superscript

`xloc:` where the x-origin is placed

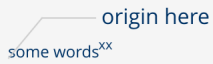
`yloc:` where the y-origin is placed

examples

write words, with a superscript choices

```
wordSuper(thisLabel,thisSuperScript,xloc,yloc)

wordSuper("some words","xx" 200,230)
```



wordSub

primitive and parameters

`wordSub(thisLabel,thisSuperScript,xloc,yloc)`

description

parameters

`thisLabel`: the words

`thisSuperScript`: the subscript

`xloc`: where the x-origin is placed

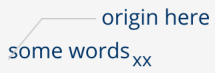
`yloc`: where the y-origin is placed

examples

write words, with a chosen subscript

```
wordSub(thisLabel,thisSubScript,xloc,yloc)

wordSub("some words","xx", 200,230)
```



symbolSub

primitive and parameters

`symbolSub(thisSymbol,thisSubScript,xloc,yloc)`

description

parameters

`thisSymbol`: a single character, to be rendered in italics

`thisSubScript`: the subscript

`xloc`: where the x-origin is placed

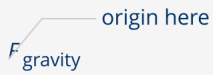
`yloc`: where the y-origin is placed

examples

write a symbol, with a subscript

```
symbolSub(thisSymbol,thisSubScript,xloc,yloc)

symbolSub("F","gravity", 200,230)
```



symbolUnit

primitive and parameters

`symbolUnit(thisSymbol,thisUnit,xloc,yloc)`

description

parameters

`thisSymbol`: the symbol

`thisUnit`: the unit

`xloc`: where the x-origin is placed

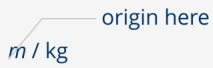
`yloc`: where the y-origin is placed

examples

write a symbol and a unit

```
symbolUnit(thisSymbol,thisUnit,xloc,yloc)

symbolUnit("m","kg", 200,230)
```



symbolSuper

primitive and parameters

`symbolSuper(thisSymbol,thisSuperScript,xloc,yloc)`

description

parameters

`thisSymbol:` a single character, to be rendered in italics

`thisSuperScript:` the set of characters to appear as a superscript

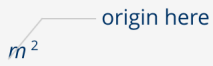
`xloc`: where the x-origin is placed

`yloc`: where the y-origin is placed

examples

write a symbol, with a superscript

```
symbolSuper(thisSymbol,thisSuperScript,xloc,yloc
symbolSuper("m","2", 200,230)
```



wordFraction

`wordFraction(topFraction,bottomFraction,xloc,yloc)`

primitive and parameters

description

parameters

`topFraction`: words to appear on top

`bottomFraction`: words to appear underneath

`xloc`: where the x-origin is placed

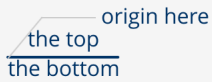
`yloc`: where the y-origin is placed

examples

write a word fraction

```
wordFraction(topFraction,bottomFraction,xloc,yloc)

wordFraction("the top","the bottom", 200,230)
```



fractionABCWrite

primitive and parameters

`fractionABCWrite(theAnswer,topFraction,bottomFraction,xloc, yloc)`

description

parameters

`theAnswer`: the answer

`topFraction`: the numerator

`bottomFraction`: the denominator

`xloc`: where the x-origin is placed

`yloc`: where the y-origin is placed

examples

write a relationship

```
fractionABCWrite(theAnswer,topFraction,bottomFraction,xloc, yloc)

fractionABCWrite("answer","top","bottom", 200,230)
```

fractionCBAWrite

`fractionCBAWrite(theAnswer,topFraction,bottomFraction,xloc,yloc)`

primitive and parameters

description

parameters

`theAnswer`: the answer

`topFraction`: the numerator

`bottomFraction`: the denominator

`xloc`: where the x-origin is placed

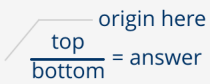
`yloc`: where the y-origin is placed

examples

write a relationship

```
fractionCBAWrite(theAnswer,topFraction,bottomFraction,xloc,yloc)

fractionCBAWrite("answer","top","bottom", 200,230)
```



title

primitive and parameters

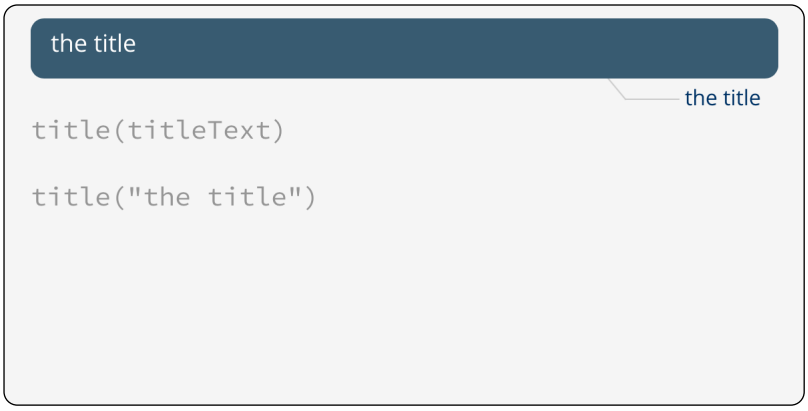
`title(titleText)`

description

parameters

`titleText`: the title text

examples



titleBold

primitive and parameters

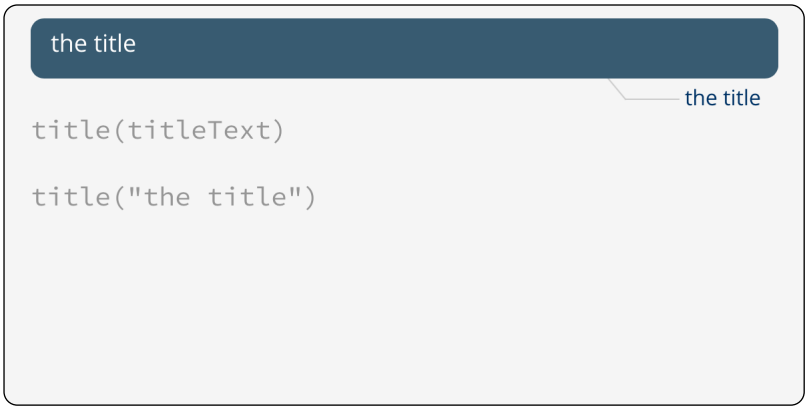
`titleBold(titleText)`

description

parameters

`titleText`: the title text, but bold

examples



comment

primitive and parameters

`comment(speakThis,xloc,yloc)`

description

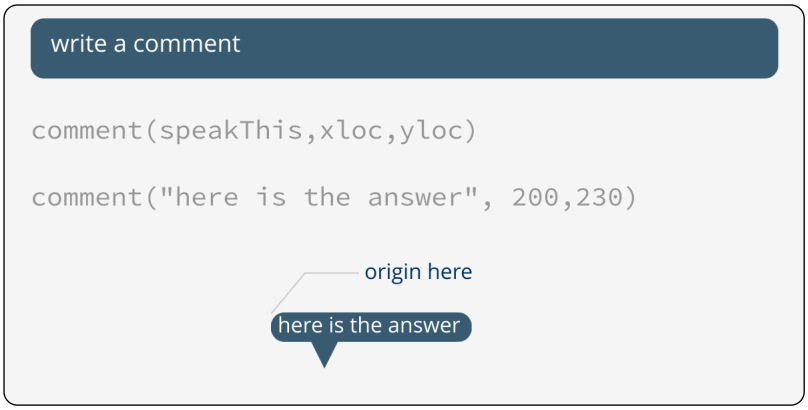
parameters

`speakThis`: what is said

`xloc`: where the x-origin is placed

`yloc`: where the y-origin is placed

examples



meter

primitive and parameters

`meter(value,precision,unit,xloc,yloc)`

description

parameters

`value`: the value

`precision`: the precision to which to express the value(decimal places)

`unit`: the unit

`xloc`: where the x-origin is placed

`yloc`: where the y-origin is placed

examples

meterinteger

primitive and parameters

`meterinteger(value,unit,xloc,yloc)`

description

parameters

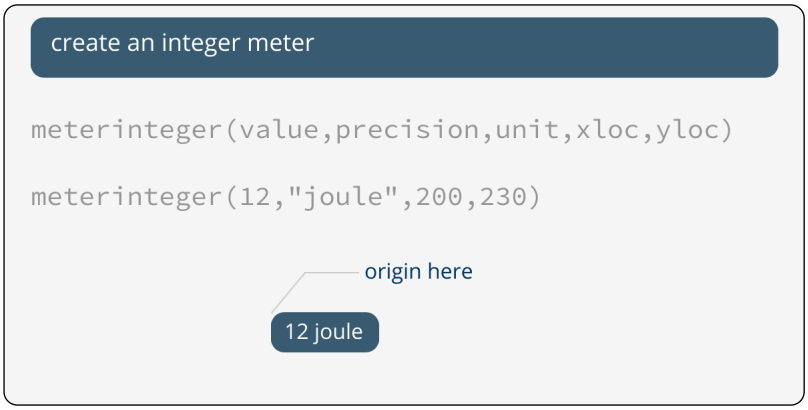
`value`: the value, integers only

`unit`: the unit

`xloc`: where the x-origin is placed

`yloc`: where the y-origin is placed

examples



leadlinediagram

primitive and parameters

`leadlinediagram(thecomment,xloc,yloc)`

description

parameters

`thecomment`: the comment

`xloc`: where the x-origin is placed

`yloc`: where the y-origin is placed

examples

one of two lead lines

```
leadlinediagram(thecomment,xloc,yloc)

leadlinediagram("A lead line", 40,280)
```

leadlinediagramnegative

primitive and parameters

`leadlinediagramnegative(thecomment, xloc, yloc)`

description

parameters

`thecomment`: the comment

`xloc`: where the x-origin is placed

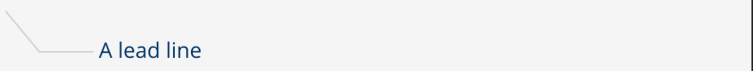
`yloc`: where the y-origin is placed

examples

one of two lead lines

```
leadlinediagramnegative(thecomment,xloc,yloc)

leadlinediagramnegative("A lead line", 40,220)
```



useful universal constants

```
var seriesoffset=107;
var loopoffset=215;
var pixelscaling = 10;
var pxscale=10;
```

FYI arrays for pspace1D and pspace2D

```
var exploredvaluesX = new Array(1);
var exploredvaluesY = new Array(1);
var exploredvalues = new Array(1);
var sparkline = new Array(0);
```

Standard colours

cdesignbackground

cpssbackground

csliderlightGrey

csliderdarkGrey

cWater

cWhite

cBlack

cItemOverviewBlue

cideaGrey

cideaBlue

cideaGreen

cideaRed

cideaBrown

cpathlinks

cstoresdarkBlue

cstoresOrange

cstoreslightOrange

cstoreslighterOrange

cstorespaleBlue

cstoresdarkRed

cstoresmidBlue

cstoresdarkGrey

cstoresmidGrey

cstoreslightGrey

cstoresdarkerGrey

cstoresdarkerRed

cstoreslightRed

cstoreframeGrey

cstoreframelightGrey

cstoreslightBlue

cstoresmiddlingBlue

cstoresgreyBlue

cstoresnuclearYellow

cstoresnuclearOrange

cstoresnuclearGrey

cstoresnuclearBlue

cstoresvibrationlightgreyBlue

cstoresvibrationdarkgreyBlue

cstoreskineticGrey

cstorethermaldarkBlue

cstorethermallightBlue

cdrawChildren

cdrawTiFo

TiFoPencil

cdrawGuides

cdrawvehicles

clabeltext

cspeechBubbleBG

cInvertedTextBG

cInvertedText

clight

csound

cabsorb

csoundsource

cdolikeme

ccircuitRed

ccircuitBlack

ccircuitloopGrey

cresistance

ccurrent

cpotentialdifference

cpendulum

cenergy

cpathwayBlue

cdeviceBlue

cdeviceLightGrey

cdeviceGrey

cboxLightGrey

cgravity

ccompression

cbuoyancy

ctension

celectric

cmagnetic

cgrip

cslip

cdrag

cgeneric

cgravity3D

ccompression3D

cbuoyancy3D

ctension3D

celectric3D

cmagnetic3D

cgrip3D

cslip3D

cdrag3D

cgeneric3D

cforceBlue

cforceGreen

cforceRed

cEfield

cBfield

cgField

cBfieldspace

cgFieldspace

cEfieldspace

cpovAlice

cpovBob

cpovCharlie

cpovSomebody

cmass

cacceleration

cmomentum

cdurationclockBlue

cdurationclockYellow

cpulleylightGrey

cpulleydarkGrey

cpulleyropeRed

cpressure0

cpressure1

cpressure2

cpressure3

cpressure4

ccongreen

cconpink

cconorange

cconlightgreen

ccongray

cconpurple

cconcyan

cabsorbpoor

cabsorbgood

cabsorbbetter

cabsorbperfect

ctransparent

cmirrorbacking

cmirrorfront

copitcallines

CeyeGrey

cmagnetpermananet

ccoilred

cmagnetcompassouter

cmagnetcompassRed

cmagnetcompassBlue

cpaneGreen

cpaneBlue

cexploreplot

cplotBG

cplotaxes

cDIPbg

cleadline

cpaneBadgecamera

cpaneBadgehands

cpaneBadgelens

cpaneBadgehighlight

cpaneBadgeblocks

cpaneBadgeedges

ctransitionBadgesWhite

ctransitionBadgesBlueBG

ctransitionBadgesBluedetail

ctransitionBadgesGrey

control_outline

control_fill

control_hover

control_background

control_press

cSIDbg

primitive and parameters

description

parameters

examples

draw a quantitative representation of a quantity of energy

```
energy(value)
```

```
energy(6);
```