Conspiracy 2htdp Library

August 24, 2022

Conspiracy provides object-oriented variations of 2htdp/image functions.

1 Object Image

The object-image module of Conspiracy implements object equivalents of Racket 2htdp/image.

```
> (require conspiracy/object-image)
'Object
'ε-Nothing
'Nothing
'ι-Container
'ι-Rectangular
'ι-Circular
'ι-Line
'ι-Linear
'ι-Linear/pulls
'ι-Regular-Polygon/angle
'ι-Regular-Polygon/angle/angle
'ι-2D-Point
'ι-Offsets
'ι-Text
'ι-Text/font
'ι-Polygon
'ι-Regular-Polygon/count
'ι-Regular-Polygon/pulls
'ι-Triangle/sss
'ι-Triangle/ass
'ι-Triangle/sas
'ι-Right-Triangle
'ι-Triangle/ssa
'ι-Triangle/aas
'ι-Triangle/asa
'ι-Triangle/saa
'ι-Star-Polygon
'ι-Radial-Star
```

```
'ι-Places
'ι-Posns
'Image
'Container
'Empty-Image
'Empty-Scene
'Ellipse
'Circle
'Line
'Add-Line
'Scene+Line
'Add-Curve
'Scene+Curve
'Add-Solid-Curve
'Text
'Text/font
'Polygon
'Regular-Polygon
'Pulled-Regular-Polygon
'Triangular
'Triangle/sss
'Triangle
'Triangle/ass
'Triangle/sas
'Isosceles-Triangle
'Right-Triangle
'Triangle/ssa
'Triangle/aas
'Triangle/asa
'Triangle/saa
'Rectangle
'Rhombus
'Square
'Star
'Star-polygon
'Radial-Star
'Add-Polygon
'Scene+Polygon
'Overlay
'Underlay
'Place-Image
'Place-Images
'Overlay/align
'Underlay/align
'Place-Image/align
'Place-Images/align
```

```
'Overlay/offset
'Underlay/offset
'Overlay/align/offset
'Underlay/align/offset
'Overlay/xy
'Underlay/xy
'Overlay/pinhole
'Underlay/pinhole
'Beside
'Above
'Beside/align
'Above/align
```

1.1 Rendering Images

Racket's 2htdp/image library provides a number of image construction functions and combinators for combining images. Conspiracy encapsulates those functions into object equivalents.

You cannot render one of the library images directly. These definitions have been constructed as 'templates' from which functional objects can be derived.

For instance:

```
> (@ show Circle)
(% Circle
        (2htdp-image-render #<procedure:...nspiracy/object.rkt:266:42>)
        (characteristics! #<procedure:...nspiracy/object.rkt:266:42>)
        (child Nothing)
        (flags (immutable no-assert))
        (implements (ι-Circular ~ι-Rectangular))
        (kinds (Ellipse))
        (parent Nothing)
        (sibling Ellipse))

• Circle template requirements:
        (radius #<flat-contract: (and/c real? (not/c negative?))>)
• Circle is a template and cannot be rendered.
```

1.1.1 Templates

In Conspiracy a template is created when an object:

- Is immutable.
- `implements` *ι-props* interfaces that require the existence of specified properties
- Does not satisfy the requirements of an *ι-porops* interface.
- `flags` no-assert

Example:

```
> (% ι-Circular (flags immutable)
   (radius (and/c real? (not/c negative?))))
'ι-Circular
> (% Circle (flags immutable no-assert)
   (implements ι-Circular))
'Circle
> (@ show Circle)
(% Circle
        (flags (immutable no-assert))
        (implements (ι-Circular))
        (kinds (Object)))

• Circle template requirements:
        (radius #<flat-contract: (and/c real? (not/c negative?))>)
```

Using the τ template syntax (which incidentally is modelled slightly on Smalltalk's syntax), you can create an instance derived from the Circle object.

Example:

```
> (@ show (τ Circle radius: 40))
(% obj#1636889
        (kinds (Circle))
        (radius 40))
```

1.1.2 Characteristics

In Conspiracy, Characteristics are those properties that make an object interesting. This is accomplished by defining an object with the following conditions:

- Is a template.
- Inherits from the Characteristics object.
- `implements` *μ-props* objects whose properties are inserted into the object's directly-defined property list. The μ-props object's propeties become the characteristics of interest.

We can extend our Circle template above, giving it characteristics as follows:

```
> (% μ-Circular (flags μ-props)
   (area () (-> real?)
         (aux radius)
         (* pi (sqr radius)))
   (circumference () (-> real?)
                  (aux radius)
                  (* 2 pi radius))
   (diameter () (-> real?)
             (aux radius)
             (* 2 radius)))
```

```
'μ-Circular
> (% Circle (kinds Characteristics) (flags immutable no-assert)
   (implements ι-Circular μ-Circular))
'Circle
> (@ show (τ Circle radius: 40))
(% obj#3028974
        (area 5026.548246)
        (circumference 251.327412)
        (diameter 80)
        (flags (immutable))
        (kinds (Circle))
        (radius 40))

• obj#3028974 characteristics requirements:
        (radius 40)
• obj#3028974 characteristics:
        (area 5026.548246)
        (circumference 251.327412)
        (diameter 80)
```

The object-image library implements its basic geometric objects with characteristics that are of interest from a geometrical perspective. What follows demonstrates the objects' rendering capabilities and does. Replacing `render` with `show` would produce results similar to those above examples, but with a rendering as well. Incidentally, the `#:precision` keyword in the example below limits the display (not the calculation) to 2 decimal places.

Example:

```
> (@ show (τ Ellipse width: 60 height: 40) #:precision '(= 2))
(% obj#4214030
        (area 3769.91)
        (child Nothing)
        (eccentricity 0.75)
        (flags (immutable))
        (foci (#(struct:posn -22.360679774997898 0) #(struct:posn 22.360679774997898
0)))
        (height 40)
        (kinds (Ellipse))
        (parent Nothing)
        (sibling Above/align)
        (width 60))

• obj#4214030 characteristics requirements:
        (height 40)
        (width 60)
• obj#4214030 characteristics:
        (area 3769.91)
        (eccentricity 0.75)
```

```
        (foci (#(struct:posn -22.360679774997898 0) #(struct:posn 22.360679774997898
0)))
```

- Rendering:

2 Object Images

The images in this file are using the Conspiracy τ syntax form for object registration.

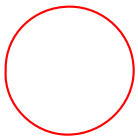
2.1 Basic Images

2.1.1 Circle

Constructs a circle with the given radius, mode, and color.

Examples:

```
> (@ render (τ Circle
               radius: 30
               outline-mode: 'outline
               pen-or-color: "red"))
```



```
> (@ render (τ Circle
               radius: 20
               outline-mode: "solid"
               pen-or-color: "blue"))
```

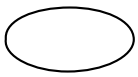
```
> (@ render (τ Circle
               radius: 20
               outline-mode: 100
               pen-or-color: "blue"))
```

2.1.2 Ellipse

Constructs an ellipse with the given width, height, mode, and color.

Examples:

```
> (@ render (τ Ellipse
               width: 60
               height: 30
               outline-mode: 'outline
               pen-or-color: "black"))
```



```
> (@ render (τ Ellipse
               width: 30
               height: 60
               outline-mode: "solid"
               pen-or-color: "blue"))
```

```
> (@ render (τ Ellipse
               width: 30
               height: 60
               outline-mode: 100
               pen-or-color: "blue"))
```

2.1.3 Line

Constructs an image representing a line segment that connects the points (0,0) to (x1,y1).

Examples

```
> (@ render (τ Line
               x1: 30
               y1: 30
               pen-or-color: "black"))
```

```
> (@ render (τ Line
               x1: 30
               y1: 20
               pen-or-color: "red"))
```

```
> (@ render (τ Line
               x1: 30
```

```
                y1: -20
                pen-or-color: "red"))
```

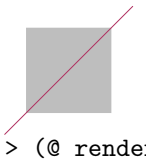
2.1.4 Add-Line

Adds a line to the image *image*, starting from the point (x1,y1) and going to the point (x2,y2). Unlike `Scene+Line`, if the line passes outside of image, the image gets larger to accommodate the line.

Examples

```
> (@ render (τ Add-Line
                x1: 0
                y1: 40
                x2: 40
                y2: 0
                pen-or-color: "maroon"
                contents:
                    (list (τ Ellipse
                             width: 40
                             height: 40
                             outline-mode: "outline"
                             pen-or-color: "maroon"))))
```

```
> (@ render (τ Add-Line
                x1: -10
                y1: 50
                x2: 50
                y2: -10
                pen-or-color: "maroon"
                contents:
                   (list (τ Rectangle
                            width: 40
                            height: 40
                            outline-mode: "solid"
                            pen-or-color: "gray"))))
```



```
> (@ render (τ Add-Line
                x1: 25
                y1: 25
                x2: 75
```

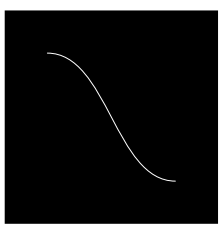
```
                y2: 75
                pen-or-color: (make-pen "goldenrod" 30 "solid" "round"
"round")
                contents: (list (τ Rectangle
                                    width: 100
                                    height: 100
                                    outline-mode: "solid"
                                    pen-or-color: "darkolivegreen"))))
```

2.1.5 Add-Curve

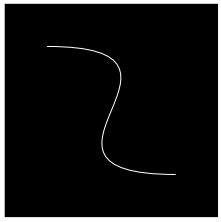
Adds a curve to image, starting at the point (x1,y1), and ending at the point (x2,y2).

Examples

```
> (@ render (τ Add-Curve
               x1: 20 y1: 20 angle1: 0 pull1: 1/3
               x2: 80 y2: 80 angle2: 0 pull2: 1/3
               pen-or-color: "white"
               contents: (list (τ Rectangle
                                  width: 100 height: 100
                                  outline-mode: "solid"
                                  pen-or-color: "black"))))
```

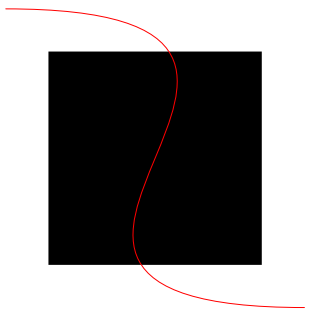


```
> (@ render (τ Add-Curve
                x1: 20 y1: 20 angle1: 0 pull1: 1
                x2: 80 y2: 80 angle2: 0 pull2: 1
                pen-or-color: "White"
                contents: (list (τ Rectangle
                                   width: 100 height: 100
                                   outline-mode: "solid"
                                   pen-or-color: "black"))))
```



```
> (@ render (τ Add-Curve
               contents: (list (τ Add-Curve
                                   contents: (list (τ Rectangle
                                                      width: 40 height:
100
                                                      outline-mode:
"solid"
                                                      pen-or-color:
"black"))
                                   x1: 20 y1: 10 angle1: 180 pull1:
1/2
                                   x2: 20 y2: 90 angle2: 180 pull2:
1/2
                                   pen-or-color: (make-pen "white" 4
"solid" "round" "round")))
               x1: 20 y1: 10 angle1: 0 pull1: 1/2
               x2: 20 y2: 90 angle2: 0 pull2: 1/2
               pen-or-color: (make-pen "white" 4 "solid" "round" "round")))
```

```
> (@ render (τ Add-Curve
               contents: (list (τ Rectangle width: 100 height: 100
                                   outline-mode: "solid"
                                   pen-or-color: "black"))
               x1: -20 y1: -20 angle1: 0 pull1: 1
               x2: 120 y2: 120 angle2: 0 pull2: 1
               pen-or-color: "red"))
```

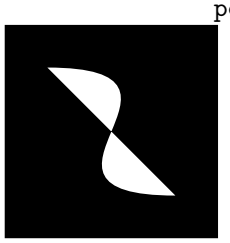


2.1.6 Add-Solid-Curve

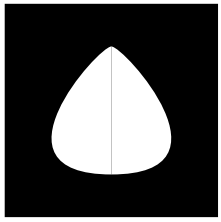
Adds a curve to image like add-curve, except it fills in the region inside the curve.

Examples:

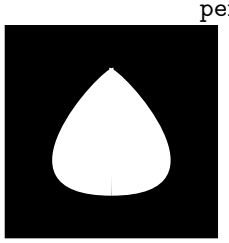
```
> (@ render (τ Add-Solid-Curve
               contents: (list (τ Rectangle width: 100 height: 100
                                  outline-mode: "solid" pen-or-color:
"black"))
               x1: 20 y1: 20 angle1: 0 pull1: 1
               x2: 80 y2: 80 angle2: 0 pull2: 1
               pen-or-color: "white"))
```



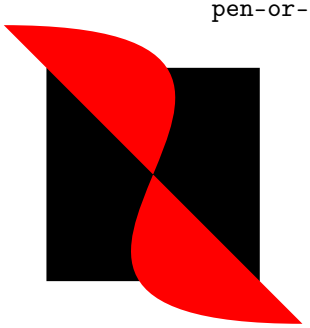
```
> (@ render (τ Add-Solid-Curve
               contents: (list (τ Add-Solid-Curve
                                  contents: (list (τ Rectangle width:
100 height: 100
                                                     outline-mode:
"solid"
                                                     pen-or-color:
"black"))
                                  x1: 50 y1: 20 angle1: 180 pull1:
1/10
                                  x2: 50 y2: 80 angle2: 0   pull2:
1
                                  pen-or-color: "white"))
               x1: 50 y1: 20 angle1: 0   pull1: 1/10
               x2: 50 y2: 80 angle2: 180 pull2: 1
               pen-or-color: "white"))
```



```
> (@ render (τ Add-Solid-Curve
               contents: (list (τ Add-Solid-Curve
                                   contents: (list (τ Rectangle width:
100 height: 100
                                                     outline-mode:
"solid"
                                                     pen-or-color:
"black"))
                                  x1: 51 y1: 20 angle1: 180 pull1: 1/10
                                  x2: 50 y2: 80 angle2: 0   pull2: 1
                                  pen-or-color: "white"))
              x1: 49 y1: 20 angle1: 0   pull1: 1/10
              x2: 50 y2: 80 angle2: 180 pull2: 1
              pen-or-color: "white"))
```



```
> (@ render (τ Add-Solid-Curve
               contents: (list (τ Rectangle width: 100 height: 100
                                   outline-mode: "solid" pen-or-color:
"black"))
               x1: -20 y1: -20 angle1: 0 pull1: 1
               x2: 120 y2: 120 angle2: 0 pull2: 1
               pen-or-color: "red"))
```



2.1.7 Text

Constructs an image that draws the given string, using the font size and color.

Examples:

```
> (@ render (τ Text text-string: "Hello" font-size: 24 color: "olive"))
```

Hello

```
> (@ render (τ Text text-string: "Goodbye" font-size: 36 color: "in-
digo"))
```

Goodbye

```
> (@ render (τ Text text-string: "Hello and\nGoodbye" font-size: 24
color: "orange"))
```

Hello and
Goodbye

2.1.8 Text/font

Constructs an image that draws the given string, using a complete font specification.

Examples:

```
> (@ render (τ Text/font
               text-string: "Hello"
               font-size: 24
               color: "olive"
               face: "Gill Sans"
               family: swiss
               style: normal
               weight: 'bold
               underline?: #f))
```

Hello

```
> (@ render (τ Text/font
               text-string: "Goodbye"
               font-size: 18
               color: "indigo"
               face: #f
               family: modern
               style: 'italic
               weight: normal
               underline?: #f))
```

Goodbye

```
> (@ render (τ Text/font
               text-string: "not really a link"
               font-size: 18
               color: "blue"
```

```
                face: #f
                family: 'roman
                style: 'normal
                weight: 'normal
                underline?: #t))
```

not really a link

2.1.9 Empty Image

The empty image. Its width and height are both zero and it does not draw at all.

Examples:

```
> (number->string (image-width (@ render (τ Empty-Image))))
0
```

2.2 Polygons

2.2.1 Triangle

Constructs a upward-pointing equilateral triangle. The side argument determines the length of the side of the triangle.

Examples:

```
> (@ render (τ Triangle side: 40 outline-mode: "solid" pen-or-color:
"tan"))
```

2.2.2 Right Triangle

Constructs a triangle with a right angle where the two sides adjacent to the right angle have lengths side1 and side2.

Examples:

```
> (@ render (τ Right-Triangle
                a: 36 b: 48
```

```
                outline-mode: "solid" pen-or-color: "black"))
```

2.2.3 Isosceles-Triangle

Creates a triangle with two equal-length sides, of length side where the angle between those sides is angle. The third leg is straight, horizontally. If the angle is less than 180, then the triangle will point up and if the angle is more, then the triangle will point down.

Examples:

```
> (@ render (τ Isosceles-Triangle
               side: 200 angle: 170
               outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Isosceles-Triangle
               side: 60 angle: 30
               outline-mode: "solid" pen-or-color: "aquamarine"))
```

```
> (@ render (τ Isosceles-Triangle
               side: 60 angle: 330
               outline-mode: "solid" pen-or-color: "lightseagreen"))
```

2.2.4 Triangle/sss

Creates a triangle where the side lengths a, b, and, c are given by a, b, and, c respectively.

Examples:

```
> (@ render (τ Triangle/sss a: 40 b: 60 c: 80
               outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Triangle/sss a: 80 b: 40 c: 60
               outline-mode: "solid" pen-or-color: "aquamarine"))
```

```
> (@ render (τ Triangle/sss a: 80 b: 80 c: 40
               outline-mode: "solid" pen-or-color: "lightseagreen"))
```

2.2.5 Triangle/ass

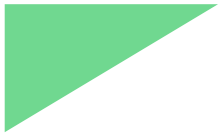
Creates a triangle where the angle A and side length a and b, are given by angle-a, b, and, c respectively. See above for a diagram showing where which sides and which

angles are which.

Examples:

```
> (@ render (τ Triangle/ass α: 10  b: 60 c: 100
                outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Triangle/ass α: 90  b: 60 c: 100
                outline-mode: "solid" pen-or-color: "aquamarine"))
```



```
> (@ render (τ Triangle/ass α: 130 b: 60 c: 100
                outline-mode: "solid" pen-or-color: "lightseagreen"))
```

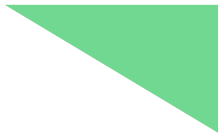
2.2.6 Triangle/sas

Creates a triangle where the side length a, angle B, and, side length c given by a, angle-b, and, c respectively. See above for a diagram showing where which sides and which angles are which.

Examples:

```
> (@ render (τ Triangle/sas a: 60  β: 10 c: 100
                outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Triangle/sas a: 60  β: 90 c: 100
                outline-mode: "solid" pen-or-color: "aquamarine"))
```



```
> (@ render (τ Triangle/sas a: 60 β: 130 c: 100
                outline-mode: "solid" pen-or-color: "lightseagreen"))
```

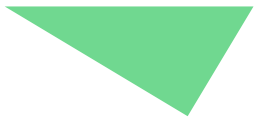
2.2.7 Triangle/ssa

Creates a triangle where the side length a, side length b, and, angle c given by a, b, and, angle-c respectively. See above for a diagram showing where which sides and which angles are which.

Examples:

```
> (@ render (τ Triangle/ssa a: 60 b: 100  γ: 10
                outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Triangle/ssa a: 60 b: 100  γ: 90
                outline-mode: "solid" pen-or-color: "aquamarine"))
```



```
> (@ render (τ Triangle/ssa a: 60 b: 100  γ: 130
                outline-mode: "solid" pen-or-color: "lightseagreen"))
```

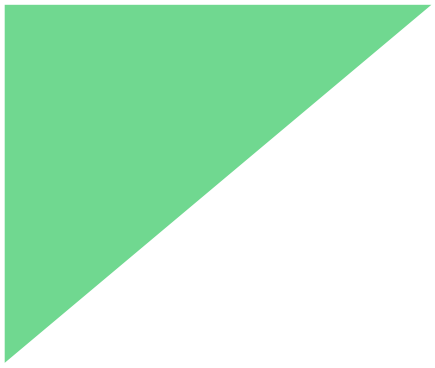
2.2.8 Triangle/aas

Creates a triangle where the angle A, angle B, and, side length c given by angle-a, angle-b, and, c respectively. See above for a diagram showing where which sides and which angles are which.

Examples:

```
> (@ render (τ Triangle/aas α: 10 β: 40  c: 200
                outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Triangle/aas α: 90 β: 40  c: 200
                outline-mode: "solid" pen-or-color: "aquamarine"))
```



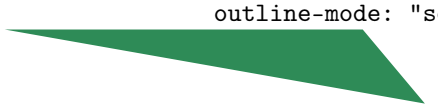
```
> (@ render (τ Triangle/aas α: 130 β: 40  c: 40
                outline-mode: "solid" pen-or-color: "lightseagreen"))
```

2.2.9 Triangle/asa

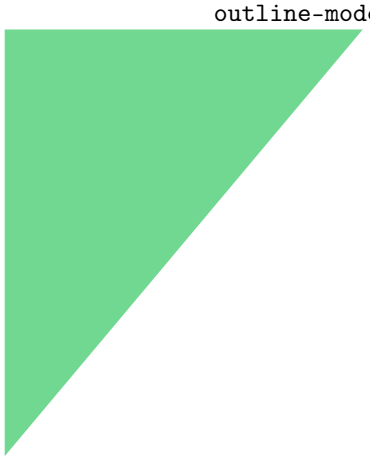
Creates a triangle where the angle A, side length b, and, angle C given by angle-a, b, and, angle-c respectively. See above for a diagram showing where which sides and which angles are which.

Examples:

```
> (@ render (τ Triangle/asa α: 10 b: 200 γ: 40
                outline-mode: "solid" pen-or-color: "seagreen"))
```



```
> (@ render (τ Triangle/asa α: 90 b: 200 γ: 40
                outline-mode: "solid" pen-or-color: "aquamarine"))
```



```
> (@ render (τ Triangle/asa α: 130 b: 40 γ: 40
                outline-mode: "solid" pen-or-color: "lightseagreen"))
```

2.2.10 Triangle/saa

Creates a triangle where the side length a, angle B, and, angle C given by a, angle-b, and, angle-c respectively. See above for a diagram showing where which sides and which angles are which.

Examples:

```
> (@ render (τ Triangle/saa a: 200 β: 10 γ: 40
                outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Triangle/saa a: 200 β: 90 γ: 40
                outline-mode: "solid" pen-or-color: "aquamarine"))
```

```
> (@ render (τ Triangle/saa a: 40 β: 130 γ: 40
                outline-mode: "solid" pen-or-color: "lightseagreen"))
```

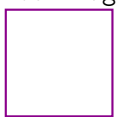
2.2.11 Square

Constructs a square.

Examples:

```
> (@ render (τ Square side: 40 outline-mode: "solid" pen-or-color:
"slateblue"))
```

```
> (@ render (τ Square side: 50 outline-mode: "outline" pen-or-color:
"darkmagenta"))
```



2.2.12 Rectangle

Constructs a rectangle with the given width, height, mode, and color.

Examples:

```
> (@ render (τ Rectangle width: 40 height: 20
               outline-mode: "outline" pen-or-color: "black"))
```

```
> (@ render (τ Rectangle width: 20 height: 40
               outline-mode: "solid" pen-or-color: "blue"))
```

2.2.13 Rhombus

Constructs a four sided polygon with all equal sides and thus where opposite angles are equal to each other. The top and bottom pair of angles is angle and the left and right are (- 180 angle).

Examples:

```
> (@ render (τ Rhombus side: 40 angle: 45
               outline-mode: "solid" pen-or-color: "magenta"))
```

```
> (@ render (τ Rhombus side: 80 angle: 150
               outline-mode: "solid" pen-or-color: "mediumpurple"))             outline-
mode:
```

2.2.14 Star

Constructs a star with five points. The side argument determines the side length of the enclosing pentagon.

Examples:

```
> (@ render (τ Star side: 40 outline-mode: "solid" pen-or-color: "gray"))
```

}|

2.2.15 Star-Polygon

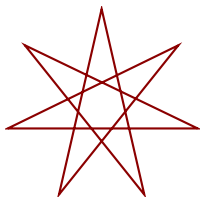
Constructs an arbitrary regular star polygon (a generalization of the regular polygons). The polygon is enclosed by a regular polygon with count sides each side long. The

polygon is actually constructed by going from vertex to vertex around the regular polygon, but connecting every step-count-th vertex (i.e., skipping every (- step-count 1) vertices).

Examples:

```
> (@ render (τ Star-Polygon side: 40 side-count: 5 step-count: 2
                outline-mode: "solid" pen-or-color: "seagreen"))
```

```
> (@ render (τ Star-Polygon side: 40 side-count:7 step-count: 3
                outline-mode: "outline" pen-or-color: "darkred"))
```



```
> (@ render (τ Star-Polygon side: 20 side-count: 10 step-count: 3
                outline-mode: "solid" pen-or-color: "cornflowerblue"))
```

2.2.16 Radial-Star

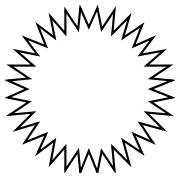
Constructs a star-like polygon where the star is specified by two radii and a number of points. The first radius determines where the points begin, the second determines where they end, and the point-count argument determines how many points the star has.

Examples:

```
> (@ render (τ Radial-Star point-count: 8 inner-radius: 8 outer-radius:
64
                outline-mode: "solid" pen-or-color: "darkslategray"))
```

```
> (@ render (τ Radial-Star point-count: 32 inner-radius: 30 outer-
radius: 40
```

```
                outline-mode: "outline" pen-or-color: "black"))
```

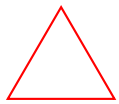


2.2.17 Regular-Polygon

Constructs a regular polygon with count sides.

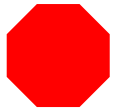
Examples:

```
> (@ render (τ Regular-Polygon side: 50 side-count: 3
                outline-mode: "outline" pen-or-color: "red"))
```



```
> (@ render (τ Regular-Polygon side: 40 side-count: 4
                outline-mode: "outline" pen-or-color: "blue"))
```

```
> (@ render (τ Regular-Polygon side: 20 side-count: 8
                outline-mode: "solid" pen-or-color: "red"))
```

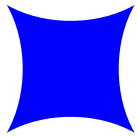


2.2.18 Pulled-Regular-Polygon

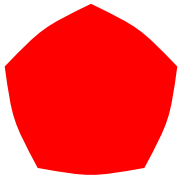
Constructs a regular polygon with count sides where each side is curved according to the pull and angle arguments. The angle argument controls the angle at which the curved version of polygon edge makes with the original edge of the polygon. Larger the pull arguments mean that the angle is preserved more at each vertex.

Examples:

```
> (@ render (τ Pulled-Regular-Polygon side: 60 side-count: 4 pull:
1/3 angle: 30
                outline-mode: "solid" pen-or-color: "blue"))
```

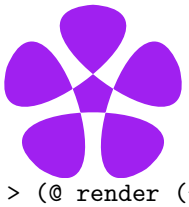


```
> (@ render (τ Pulled-Regular-Polygon side: 50 side-count: 5 pull:
1/2 angle: -10
                outline-mode: "solid" pen-or-color: "red"))
```

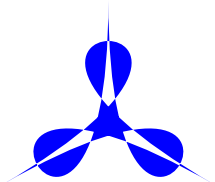


```
> (@ render (τ Pulled-Regular-Polygon side: 50 side-count: 5 pull:
1 angle: 140
               outline-mode: "solid" pen-or-color: "purple"))
```

```
> (@ render (τ Pulled-Regular-Polygon side: 50 side-count: 5 pull:
1.1 angle: 140
               outline-mode: "solid" pen-or-color: "purple"))
```



```
> (@ render (τ Pulled-Regular-Polygon side: 100 side-count: 3 pull:
1.8 angle: 30
               outline-mode: "solid" pen-or-color: "blue"))
```



2.2.19 Polygon

Constructs a polygon connecting the given vertices.

Examples:

```
> (@ render (τ Polygon
               vertices: (list (make-posn 0 0)
                               (make-posn -10 20)
                               (make-posn 60 0)
                               (make-posn -10 -20))
               outline-mode: "solid"
               pen-or-color: "burlywood"))
```

```
> (@ render (τ Polygon
               vertices: (list (make-pulled-point 1/2 20 0 0 1/2 -
20)
                   (make-posn -10 20)
                   (make-pulled-point 1/2 -20 60 0 1/2 20)
                   (make-posn -10 -20))
               outline-mode: "solid"
               pen-or-color: "burlywood"))
```

```
> (@ render (τ Polygon
               vertices:
               (make-posn 0 0)
               (make-posn 0 40)
               (make-posn 20 40)
               (make-posn 20 60)
               (make-posn 40 60)
               (make-posn 40 20)
               (make-posn 20 20)
               (make-posn 20 0)
               outline-mode: "solid"
               pen-or-color: "plum"))
```

```
> (@ render (τ Underlay
               contents:
               (τ Rectangle width: 80 height: 80
                  outline-mode: "solid" pen-or-color: "mediumseagreen")
               (τ Polygon
                  vertices:
                  (make-posn 0 0)
                  (make-posn 50 0)
                  (make-posn 0 50)
                  (make-posn 50 50)
                  outline-mode: "outline"
                  pen-or-color: (make-pen "darkslategray" 10 "solid"
"round" "round"))))
```

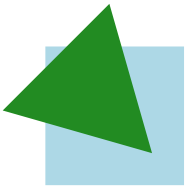
```
> (@ render (τ Underlay
               contents:
               (τ Rectangle width: 90 height: 80
                  outline-mode: "solid" pen-or-color: "mediumseagreen")
               (τ Polygon
                  vertices:
                  (make-posn 0 0)
                  (make-posn 50 0)
                  (make-posn 0 50)
                  (make-posn 50 50)
                  outline-mode: "outline"
                  pen-or-color: (make-pen "darkslategray" 10 "solid"
                                          "projecting" "miter"))))
```

2.2.20 Add-Polygon

Adds a closed polygon to the image image, with vertices as specified in posns (relative to the top-left corner of image). Unlike scene+polygon, if the polygon goes outside the bounds of image, the result is enlarged to accommodate both.

Examples:

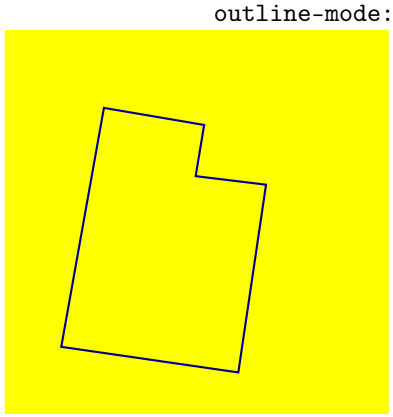
```
> (@ render (τ Add-Polygon
               contents: (list (τ Square side: 65
                                  outline-mode: "solid" pen-or-color:
"light blue"))
               vertices:
               (make-posn 30 -20)
               (make-posn 50 50)
               (make-posn -20 30)
               outline-mode: "solid" pen-or-color: "forest green"))
```



```
> (@ render (τ Add-Polygon
               contents: (list (τ Square side: 65
                                  outline-mode: "solid" pen-or-color:
"light blue"))
               vertices:
```

```
                (make-posn 30 -20)
                (make-pulled-point 1/2 30 50 50 1/2 -30)
                (make-posn -20 30)
                outline-mode: "solid" pen-or-color: "forest green"))
```

```
> (@ render (τ Add-Polygon
                contents: (list (τ Square side: 180
                                   outline-mode: "solid" pen-or-color:
"yellow"))
                vertices:
                (make-posn 109 160)
                (make-posn 26 148)
                (make-posn 46 36)
                (make-posn 93 44)
                (make-posn 89 68)
                (make-posn 122 72)
                outline-mode: "outline" pen-or-color: "dark blue"))
```



```
> (@ render (τ Add-Polygon
                contents: (list (τ Square side: 50
                                   outline-mode: "solid" pen-or-color:
"light blue"))
                vertices:
                (make-posn 25 -10)
                (make-posn 60 25)
                (make-posn 25 60)
                (make-posn -10 25)
                outline-mode: "solid" pen-or-color: "pink"))
```

2.2.21 Scene+Polygon

Adds a closed polygon to the image image, with vertices as specified in posns (relative to the top-left corner of image). Unlike add-polygon, if the polygon goes outside the bounds of image, the result is clipped to image.

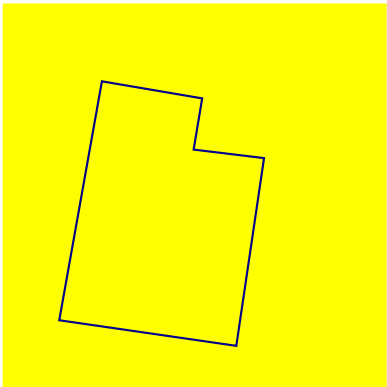
Examples:

```
> (@ render (τ Scene+Polygon
               contents: (list (τ Square side: 65
                                  outline-mode: "solid" pen-or-color:
"light blue"))
               vertices:
               (make-posn 30 -20)
               (make-posn 50 50)
               (make-posn -20 30)
               outline-mode: "solid" pen-or-color: "forest green"))
```

```
> (@ render (τ Scene+Polygon
               contents: (list (τ Square side: 65
                                  outline-mode: "solid" pen-or-color:
"light blue"))
               vertices:
               (make-posn 30 -20)
               (make-pulled-point 1/2 -30 50 50 1/2 30)
               (make-posn -20 30)
               outline-mode: "solid" pen-or-color: "forest green"))
```

```
> (@ render (τ Scene+Polygon
               contents: (list (τ Square side: 180
                                  outline-mode: "solid" pen-or-color:
"yellow"))
               vertices:
               (make-posn 109 160)
               (make-posn 26 148)
```

```
                (make-posn 46 36)
                (make-posn 93 44)
                (make-posn 89 68)
                (make-posn 122 72)
                outline-mode: "outline" pen-or-color: "dark blue"))
```



```
> (@ render (τ Scene+Polygon
                contents: (list (τ Square side: 50
                                   outline-mode: "solid" pen-or-color:
"light blue"))
                vertices:
                (make-posn 25 -10)
                (make-posn 60 25)
                (make-posn 25 60)
                (make-posn -10 25)
                outline-mode: "solid" pen-or-color: "pink"))
```


2.3 Overlaying Images

2.3.1 Overlay

Overlays all of its arguments building a single image. The first argument goes on top of the second argument, which goes on top of the third argument, etc. The images are all lined up on their centers.

Examples:

```
> (@ render (τ Overlay
                contents:
                (τ Rectangle width: 30 height: 60
                   outline-mode: "solid" pen-or-color: "orange")
                (τ Ellipse width: 60 height: 30
                   outline-mode: "solid" pen-or-color: "purple")))
```

```
> (@ render (τ Overlay
               contents:
               (τ Ellipse width: 10 height: 10
                  outline-mode: "solid" pen-or-color: "red")
               (τ Ellipse width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "black")
               (τ Ellipse width: 30 height: 30
                  outline-mode: "solid" pen-or-color: "red")
               (τ Ellipse width: 40 height: 40
                  outline-mode: "solid" pen-or-color: "black")
               (τ Ellipse width: 50 height: 50
                  outline-mode: "solid" pen-or-color: "red")
               (τ Ellipse width: 60 height: 60
                  outline-mode: "solid" pen-or-color: "black")))
```

```
> (@ render (τ Overlay
               contents:
               (τ Regular-Polygon side: 20 side-count: 5
                  outline-mode: "solid" pen-or-color: (make-color  50  50
255))
               (τ Regular-Polygon side: 26 side-count: 5
                  outline-mode: "solid" pen-or-color: (make-color 100
100 255))
               (τ Regular-Polygon side: 32 side-count: 5
                  outline-mode: "solid" pen-or-color: (make-color 150
150 255))
               (τ Regular-Polygon side: 38 side-count: 5
                  outline-mode: "solid" pen-or-color: (make-color 200
200 255))
               (τ Regular-Polygon side: 44 side-count: 5
                  outline-mode: "solid" pen-or-color: (make-color 250
250 255))))
```

2.3.2 Overlay/align

Overlays all of its image arguments, much like the overlay function, but using x-place and y-place to determine where the images are lined up. For example, if x-place and y-place are both "middle", then the images are lined up on their centers.

Examples:

```
> (@ render (τ Overlay/align x-place: "left" y-place: "middle"
               contents: (τ Rectangle width: 30 height: 60
                            outline-mode: "solid" pen-or-color: "or-
ange")
               (τ Ellipse width: 60 height: 30
                  outline-mode: "solid" pen-or-color: "purple")))
```

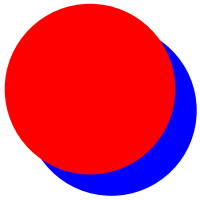
```
> (@ render (τ Overlay/align x-place: "right" y-place: "bottom"
               contents:
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "silver")
               (τ Rectangle width: 30 height: 30
                  outline-mode: "solid" pen-or-color: "seagreen")
               (τ Rectangle width: 40 height: 40
                  outline-mode: "solid" pen-or-color: "silver")
               (τ Rectangle width: 50 height: 50
                  outline-mode: "solid" pen-or-color: "seagreen")))
```

2.3.3 Overlay/offset

Just like overlay, this function lines up its image arguments on top of each other. Unlike overlay, it moves i2 by x pixels to the right and y down before overlaying them.

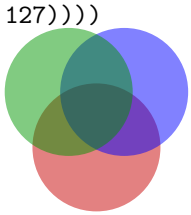
Examples:

```
> (@ render (τ Overlay/offset x-offset: 10 y-offset: 10
               contents:
               (τ Circle radius: 40
                  outline-mode: "solid" pen-or-color: "red")
               (τ Circle radius: 40
                  outline-mode: "solid" pen-or-color: "blue")))
```



```
> (@ render (τ Overlay/offset x-offset: 70 y-offset: 0
               contents:
               (τ Overlay/offset x-offset: -50 y-offset: 0
                  contents: (τ Rectangle width: 60 height: 20
                               outinel-mode: "solid" pen-or-color:
"black")
                  (τ Circle radius: 20
                     outline-mode: "solid" pen-or-color: "darkorange"))
               (τ Circle radius: 20
                  outine-mode: "solid" pen-or-color: "darkorange")))
```

```
> (@ render (τ Overlay/offset x-offset: 0 y-offset: 26
               contents:
               (τ Overlay/offset x-offset: 26 y-offset: 0
                  contents: (τ Circle radius: 30
                               outline-mode: 'solid pen-or-color: (color
0 150 0 127))
                  (τ Circle radius: 30
                     outline-mode: 'solid pen-or-color: (color 0 0
255 127)))
               (τ Circle radius: 30
                  outline-mode: 'solid pen-or-color: (color 200 0 0
127))))
```



2.3.4 Overlay/align/offset

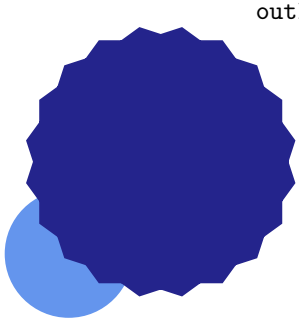
Overlays image i1 on top of i2, using x-place and y-place as the starting points for the overlaying, and then adjusts i2 by x to the right and y pixels down.

Examples:

```
> (@ render (τ Overlay/align/offset
               x-place: "right" y-place: "bottom" x-offset: 10 y-offset:
10
```

```
                contents:
                (τ Star-Polygon side: 20 side-count: 20 step-count:
3
                   outline-mode: "solid" pen-or-color: "navy")
                (τ Circle radius: 30
                   outline-mode: "solid" pen-or-color: "cornflowerblue")))
```

```
> (@ render (τ Overlay/align/offset
                x-place: "left" y-place: "bottom" x-offset: -10 y-offset:
10
                contents:
                (τ Star-Polygon side: 20 side-count: 20 step-count:
3
                   outline-mode: "solid" pen-or-color: "navy")
                (τ Circle radius: 30
                   outline-mode: "solid" pen-or-color: "cornflowerblue")))
```



2.3.5 Overlay/xy

Constructs an image by overlaying i1 on top of i2. The images are initially lined up on their upper-left corners and then i2 is shifted to the right by x pixels and down by y pixels.

Examples:

```
> (@ render (τ Overlay/xy x: 20 y: 0
                contents:
                (τ Rectangle width: 20 height: 20
                   outline-mode: "outline" pen-or-color: "black")
                (τ Rectangle width: 20 height: 20
```

```
                  outline-mode: "outline" pen-or-color: "black")))
```

```
> (@ render (τ Overlay/xy x: 10 y: 10
               contents:
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "red")
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "black")))
```

```
> (@ render (τ Overlay/xy x: -10 y: -10
               contents:
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "red")
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "black")))
```

```
> (@ render (τ Overlay/xy x: 10 y: 15
               contents:
               (τ Overlay/xy x: 20 y: 15
                  contents:
                  (τ Ellipse width: 40 height: 40
                     outline-mode: "outline" pen-or-color: "black")
                  (τ Ellipse width: 10 height: 10
                     outline-mode: "solid" pen-or-color: "forestgreen"))
               (τ Ellipse width: 10 height: 10
                  outline-mode: "solid" pen-or-color: "forestgreen")))
```

2.4 Underlaying Images

2.4.1 Underlay

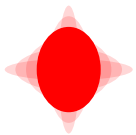
Underlays all of its arguments building a single image.

Examples:

```
> (@ render (τ Underlay
               contents:
               (τ Rectangle width: 30 height: 60
                  outline-mode: "solid" pen-or-color: "orange")
               (τ Ellipse width: 60 height: 30
                  outline-mode: "solid" pen-or-color: "purple")))
```

```
> (@ render (τ Underlay
               contents:
               (τ Ellipse width: 10 height: 60
                  outline-mode: "solid" pen-or-color: "red")
               (τ Ellipse width: 20 height: 50
                  outline-mode: "solid" pen-or-color: "black")
               (τ Ellipse width: 30 height: 40
                  outline-mode: "solid" pen-or-color: "red")
               (τ Ellipse width: 40 height: 30
                  outline-mode: "solid" pen-or-color: "black")
               (τ Ellipse width: 50 height: 20
                  outline-mode: "solid" pen-or-color: "red")
               (τ Ellipse width: 60 height: 10
                  outline-mode: "solid" pen-or-color: "black")))
```

```
> (@ render (τ Underlay
               contents:
               (τ Ellipse width: 10 height: 60
                  outline-mode: 40 pen-or-color: "red")
               (τ Ellipse width: 20 height: 50
                  outline-mode: 40 pen-or-color: "red")
               (τ Ellipse width: 30 height: 40
                  outine-mode: 40 pen-or-color: "red")
               (τ Ellipse width: 40 height: 30
                  outline-mode: 40 pen-or-color: "red")
               (τ Ellipse width: 50 height: 20
                  outline-mode: 40 pen-or-color: "red")
               (τ Ellipse width: 60 height: 10
                  outline-mode: 40 pen-or-color: "red")))
```



2.4.2 Underlay/align

Underlays all of its image arguments, much like the underlay function, but using x-place and y-place to determine where the images are lined up. For example, if x-place

and y-place are both "middle", then the images are lined up on their centers.

Examples:

```
> (@ render (τ Underlay/align x-place: "left" y-place: "middle"
               contents:
               (τ Rectangle width: 30 height: 60
                  outline-mode: "solid" pen-or-color: "orange")
               (τ Ellipse width: 60 height: 30
                  outline-mode: "solid" pen-or-color: "purple")))
```

```
> (@ render (τ Underlay/align x-place: "right" y-place: "top"
               contents:
               (τ Rectangle width: 50 height: 50
                  outline-mode: "solid" pen-or-color: "seagreen")
               (τ Rectangle width: 40 height: 40
                  outline-mode: "solid" pen-or-color: "silver")
               (τ Rectangle width: 30 height: 30
                  outline-mode: "solid" pen-or-color: "seagreen")
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "silver")))
```

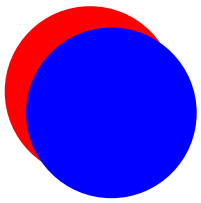
```
> (@ render (τ Underlay/align x-place: "left" y-place: "middle"
               contents:
               (τ Rectangle width: 50 height: 50
                  outline-mode: 50 pen-or-color: "seagreen")
               (τ Rectangle width: 40 height: 40
                  outline-mode: 50 pen-or-color: "seagreen")
               (τ Rectangle width: 30 height: 30
                  outline-mode: 50 pen-or-color: "seagreen")
               (τ Rectangle width: 20 height: 20
                  outline-mode: 50 pen-or-color: "seagreen")))
```

2.4.3 Underlay/offset

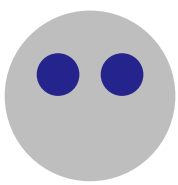
Just like underlay, this function lines up its first image argument underneath the second. Unlike underlay, it moves i2 by x pixels to the right and y down before underlaying them.

Examples:

```
> (@ render (τ Underlay/offset x-offset: 10 y-offset: 10
               contents:
               (τ Circle radius: 40
                  outline-mode: "solid" pen-or-color: "red")
               (τ Circle radius: 40
                  outline-mode: "solid" pen-or-color: "blue")))
```



```
> (@ render (τ Underlay/offset x-offset: 0 y-offset: -10
               contents:
               (τ Circle radius: 40
                  outline-mode: "solid" pen-or-color: "gray")
               (τ Underlay/offset x-offset: -30 y-offset: 0
                  contents: (τ Circle radius: 10
                               outline-mode: "solid" pen-or-color:
"navy")
                  (τ Circle radius: 10
                     outline-mode: "solid" pen-or-color: "navy"))))
```

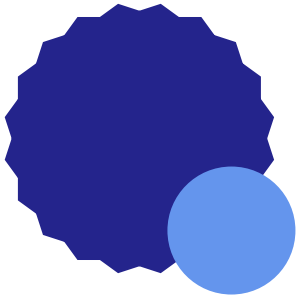


2.4.4 Underlay/align/offset

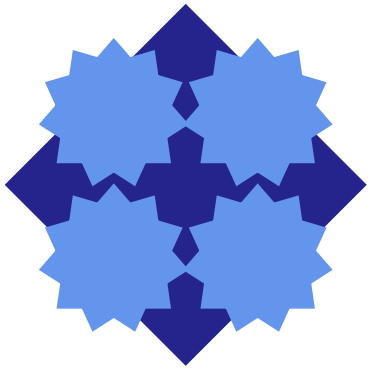
Underlays image i1 underneath i2, using x-place and y-place as the starting points for the combination, and then adjusts i2 by x to the right and y pixels down.

Examples:

```
> (@ render (τ Underlay/align/offset
               x-place: "right" y-place: "bottom" x-offset: 10 y-offset:
10
               contents:
               (τ Star-Polygon side: 20 side-count: 20 step-count:
3
                  outline-mode: "solid" pen-or-color: "navy")
               (τ Circle radius: 30
                  outline-mode: "solid" pen-or-color: "cornflowerblue")))
```



```
> (@ render (τ Underlay/align/offset
               x-place: "right" y-place: "bottom" x-offset: -16 y-
offset: -16
               contents:
               (τ Underlay/align/offset
                  x-place: "left" y-place: "bottom" x-offset: 16 y-
offset: -16
                  contents:
                  (τ Underlay/align/offset
                     x-place: "right" y-place: "top" x-offset: -16
y-offset: 16
                     contents:
                     (τ Underlay/align/offset
                        x-place: "left" y-place: "top" x-offset: 16
y-offset: 16
                        contents:
                        (τ Rhombus side: 120 angle: 90
                           outline-mode: "solid" pen-or-color: "navy")
                        (τ Star-Polygon side: 20 side-count: 11 step-
count: 3
                           outline-mode: "solid" pen-or-color: "corn-
flowerblue"))
                     (τ Star-Polygon side: 20 side-count: 11 step-
count: 3
                        outline-mode: "solid" pen-or-color: "corn-
flowerblue"))
                  (τ Star-Polygon side: 20 side-count: 11 step-count:
3
                     outline-mode: "solid" pen-or-color: "cornflowerblue"))
               (τ Star-Polygon side: 20 side-count: 11 step-count:
3
                  outline-mode: "solid" pen-or-color: "cornflowerblue")))
```



2.4.5 Unerly/xy

Constructs an image by underlaying i1 underneath i2. The images are initially lined up on their upper-left corners and then i2 is shifted to the right by x pixels to and down by y pixels.

Examples:

```
> (@ render (τ Underlay/xy x: 20 y: 0
               contents:
               (τ Rectangle width: 20 height: 20
                  outline-mode: "outline" pen-or-color: "black")
               (τ Rectangle width: 20 height: 20
                  outline-mode: "outline" pen-or-color: "black")))
```

```
> (@ render (τ Underlay/xy x: 10 y: 10
               contents:
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "red")
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "black")))
```

```
> (@ render (τ Underlay/xy x: -10 y: -10
               contents:
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "red")
               (τ Rectangle width: 20 height: 20
                  outline-mode: "solid" pen-or-color: "black")))
```

```
> (@ render (τ Underlay/xy x: 20 y: 15
               contents:
               (τ Underlay/xy x: 10 y: 15
```

```
                contents:
                (τ Ellipse width: 40 height: 40
                   outline-mode: "solid" pen-or-color: "gray")
                (τ Ellipse width: 10 height: 10
                   outline-mode: "solid" pen-or-color: "forestgreen"))
             (τ Ellipse width: 10 height: 10
                outline-mode: "solid" pen-or-color: "forestgreen")))
```

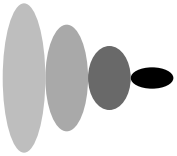
2.5 Beside Images

2.5.1 Beside

Constructs an image by placing all of the argument images in a horizontal row, aligned along their centers.

Examples

```
> (@ render (τ Beside
             contents:
             (τ Ellipse width: 20 height: 70
                ouline-mode: "solid" pen-or-color: "gray")
             (τ Ellipse width: 20 height: 50
                outline-mode: "solid" pen-or-color: "darkgray")
             (τ Ellipse width: 20 height: 30
                outline-mode: "solid" pen-or-color: "dimgray")
             (τ Ellipse width: 20 height: 10
                outline-mode: "solid" pen-or-color: "black")))
```



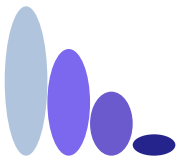
2.5.2 Beside/align

Constructs an image by placing all of the argument images in a horizontal row, lined up as indicated by the y-place argument. For example, if y-place is "middle", then the images are placed side by side with their centers lined up with each other.

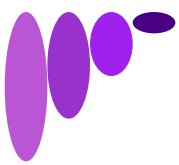
Examples:

```
> (@ render (τ Beside/align y-place: "bottom"
             contents:
             (τ Ellipse width: 20 height: 70
                outline-mode: "solid" pen-or-color: "lightsteel-
blue")
             (τ Ellipse width: 20 height: 50
```

```
                  outline-mode: "solid" pen-or-color: "mediumslate-
blue")
               (τ Ellipse width: 20 height: 30
                  outline-mode: "solid" pen-or-color: "slateblue")
               (τ Ellipse width: 20 height: 10
                  outline-mode: "solid" pen-or-color: "navy")))
```



```
> (@ render (τ Beside/align y-place: "top"
               contents:
               (τ Ellipse width: 20 height: 70
                  outline-mode: "solid" pen-or-color: "mediumorchid")
               (τ Ellipse width: 20 height: 50
                  outline-mode: "solid" pen-or-color: "darkorchid")
               (τ Ellipse width: 20 height: 30
                  outline-mode: "solid" pen-or-color: "purple")
               (τ Ellipse width: 20 height: 10
                  outline-mode: "solid" pen-or-color: "indigo")))
```



```
> (@ render (τ Beside/align y-place: "baseline"
               contents:
               (τ Text text-string: "ijy" font-size: 18 color: "black")
               (τ Text text-string: "ijy" font-size: 24 color: "black")))
```

ijyijy

2.6 Above Images

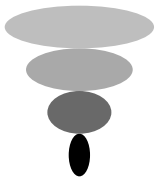
2.6.1 Above

Constructs an image by placing all of the argument images in a vertical row, aligned along their centers.

Examples:

```
> (@ render (τ Above
               contents:
               (τ Ellipse width: 70 height: 20
                  outline-mode: "solid" pen-or-color: "gray")
               (τ Ellipse width: 50 height: 20
                  outline-mode: "solid" pen-or-color: "darkgray")
               (τ Ellipse width: 30 height: 20
```

```
                 outline-mode: "solid" pen-or-color: "dimgray")
               (τ Ellipse width: 10 height: 20
                 outline-mode: "solid" pen-or-color: "black")))
```



2.6.2 Above/align

Constructs an image by placing all of the argument images in a vertical row, lined up as indicated by the x-place argument. For example, if x-place is "middle", then the images are placed above each other with their centers lined up.

Examples:

```
> (@ render (τ Above/align y-place: "right"
               contents:
               (τ Ellipse width: 70 height: 20
                  outline-mode: "solid" pen-or-color: "gold")
               (τ Ellipse width: 50 height: 20
                  outline-mode: "solid" pen-or-color: "goldenrod")
               (τ Ellipse width: 30 height: 20
                  outline-mode: "solid" pen-or-color: "darkgolden-
rod")
               (τ Ellipse width: 10 height: 20
                  outline-mode: "solid" pen-or-color: "sienna")))
```

```
> (@ render (τ Above/align y-place: "left"
              contents:
              (τ Ellipse width: 70 height: 20
                 outline-mode: "solid" pen-or-color: "yellowgreen")
              (τ Ellipse width: 50 height: 20
                 outline-mode: "solid" pen-or-color: "olivedrab")
              (τ Ellipse width: 30 height: 20
                 outline-mode: "solid" pen-or-color: "darkolivegreen")
              (τ Ellipse width: 10 height: 20
                 outline-mode: "solid" pen-or-color: "darkgreen")))
```

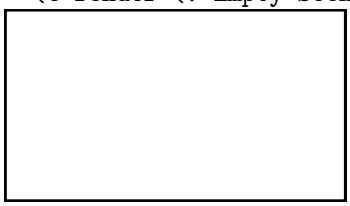
2.7 Placing Images

2.7.1 Empty-Scene

Creates an empty scene, i.e., a white rectangle with a black outline.

Examples:

```
> (@ render (τ Empty-Scene width: 160 height: 90))
```



2.7.2 Place-Image

Places image onto scene with its center at the coordinates (x,y) and crops the resulting image so that it has the same size as scene. The coordinates are relative to the top-left of scene.

Examples:

```
> (@ render (τ Place-Image x: 24 y: 24
               contents:
               (τ Triangle side: 32
                  outline-mode: "solid" pen-or-color: "red")
               (τ Rectangle width: 48 height: 48
                  outline-mode: "solid" pen-or-color: "gray")))
```

```
> (@ render (τ Place-Image x: 24 y: 24
               contents:
               (τ Triangle side: 64
                  outline-mode: "solid" pen-or-color: "red")
               (τ Rectangle width: 48 height: 48
                  outline-mode: "solid" pen-or-color: "gray")))
```

```
> (@ render (τ Place-Image x: 18 y: 20
```

```
                contents:
                (τ Circle radius: 4
                   outline-mode: "solid" pen-or-color: "white")
                (τ Place-Image x: 0 y: 6
                   contents:
                   (τ Circle radius: 4
                      outline-mode: "solid" pen-or-color: "white")
                   (τ Place-Image x: 14 y: 2
                      contents:
                      (τ Circle radius: 4
                         outline-mode: "solid" pen-or-color: "white")
                      (τ Place-Image x: 8 y: 14
                         contents:
                         (τ Circle radius: 4
                            outline-mode: "solid" pen-or-color: "white")
                         (τ Rectangle width: 24 height: 24
                            outline-mode: "solid" pen-or-color: "gold-
enrod")))))))
```

2.7.3 Place-Image/align

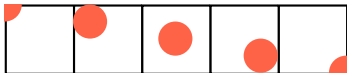
Like place-image, but uses image's x-place and y-place to anchor the image. Also, like place-image, place-image/align crops the resulting image so that it has the same size as scene.

Examples:

```
> (@ render (τ Place-Image/align
                x: 64 y: 64 x-place: "right" y-place: "bottom"
                contents:
                (τ Triangle side: 48
                   outline-mode: "solid" pen-or-color: "yellowgreen")
                (τ Rectangle width: 64 height: 64
                   outline-mode: "solid" pen-or-color: "mediumgold-
enrod")))
```

```
> (@ render (τ Beside
                contents:
                (τ Place-Image/align
                   x: 0 y: 0 x-place: "center" y-place: "center"
                   contents:
                   (τ Circle radius: 8
                      outline-mode: "solid" pen-or-color: "tomato")
```

```
   (τ Rectangle width: 32 height: 32
      outline-mode: "outline" pen-or-color: "black"))
(τ Place-Image/align
   x: 8 y: 8 x-place: "center" y-place: "center"
   contents:
   (τ Circle radius: 8
      outline-mode: "solid" pen-or-color: "tomato")
   (τ Rectangle width: 32 height: 32
      outline-mode: "outline" pen-or-color: "black"))
(τ Place-Image/align
   x: 16 y: 16 x-place: "center" y-place: "center"
   contents:
   (τ Circle radius: 8
      outline-mode: "solid" pen-or-color: "tomato")
   (τ Rectangle width: 32 height: 32
      outline-mode: "outline" pen-or-color: "black"))
(τ Place-Image/align
   x: 24 y: 24 x-place: "center" y-place: "center"
   contents:
   (τ Circle radius: 8
      outline-mode: "solid" pen-or-color: "tomato")
   (τ Rectangle width: 32 height: 32
      outline-mode: "outline" pen-or-color: "black"))
(τ Place-Image/align
   x: 32 y: 32 x-place: "center" y-place: "center"
   contents:
   (τ Circle radius: 8
       outline-mode: "solid" pen-or-color: "tomato")
   (τ Rectangle width: 32 height: 32
      outline-mode: "outline" pen-or-color: "black"))))
```



2.7.4 Place-Images

Places each of images into scene like place-image would, using the coordinates in posns as the x and y arguments to place-image.

Examples:

```
> (@ render (τ Place-Images
               posns:
               (make-posn 18 20)
               (make-posn 0 6)
               (make-posn 14 2)
               (make-posn 8 14)
               contents:
```

```
                (τ Circle radius: 4
                   outline-mode: "solid" pen-or-color: "white")
                (τ Circle radius: 4
                   outline-mode: "solid" pen-or-color: "white")
                (τ Circle radius: 4
                   outline-mode: "solid" pen-or-color: "white")
                (τ Circle radius: 4
                   outline-mode: "solid" pen-or-color: "white")
                (τ Rectangle width: 24 height: 24
                   outline-mode: "solid" pen-or-color: "goldenrod")))
```

2.7.5 Place-Images/align

Like place-images, except that it places the images with respect to x-place and y-place.

Examples:

```
> (@ render (τ Place-Images/align
                posns:
                (make-posn 64 64)
                (make-posn 64 48)
                (make-posn 64 32)
                (make-posn 64 16)
                x-place: "right" y-place: "bottom"
                contents:
                (τ Triangle side: 48
                   outline-mode: "solid" pen-or-color: "yellowgreen")
                (τ Triangle side: 48
                   ouline-mode: "solid" pen-or-color: "yellowgreen")
                (τ Triangle side: 48
                   outline-mode: "solid" pen-or-color: "yellowgreen")
                (τ Triangle side: 48
                   outline-mode: "solid" pen-or-color: "yellowgreen")
                (τ Rectangle width: 64 height: 64
                   outline-mode: "solid" pen-or-color: "mediumgold-
enrod")))
```

2.7.6 Scene+Line

Adds a line to the image scene, starting from the point (x1,y1) and going to the point (x2,y2); unlike add-line, this function crops the resulting image to the size of scene.

Examples:

```
> (@ render (τ Scene+Line x1: 0 y1: 40 x2: 40 y2: 0
                contents:
                (list (τ Ellipse width: 40 height: 40
                         outline-mode: "outline" pen-or-color: "ma-
roon"))
                pen-or-color: "maroon"))
```

```
> (@ render (τ Scene+Line x1: -10 y1: 50 x2: 50 y2: -10
                contents:
                (list (τ Rectangle width: 40 height: 40
                         outline-mode: "solid" pen-or-color: "gray"))
                pen-or-color: "maroon"))
```

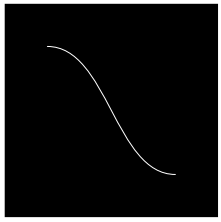
```
> (@ render (τ Scene+Line x1: 25 y1: 25 x2: 100 y2: 100
                contents:
                (list (τ Rectangle width: 100 height: 100
                         outline-mode: "solid" pen-or-color: "dark-
olivegreen"))
                pen-or-color: (make-pen "goldenrod" 30 "solid" "round"
"round")))
```

2.7.7 Scene+Curve

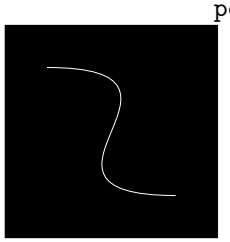
Adds a curve to scene, starting at the point (x1,y1), and ending at the point (x2,y2).

Examples:

```
> (@ render (τ Scene+Curve
                x1: 20 y1: 20 angle1: 0 pull1: 1/3
                x2: 80 y2: 80 angle2: 0 pull2: 1/3
                contents:
                (list (τ Rectangle width: 100 height: 100
                         outline-mode: "solid" pen-or-color: "black"))
                pen-or-color: "white"))
```



```
> (@ render (τ Scene+Curve
               x1: 20 y1: 20 angle1: 0 pull1: 1
               x2: 80 y2: 80 angle2: 0 pull2: 1
               contents:
               (list (τ Rectangle width: 100 height: 100
                        outline-mode: "solid" pen-or-color: "black"))
               pen-or-color: "white"))
```



```
> (@ render (τ Scene+Curve
               x1: 20 y1: 10 angle1: 0 pull1: 1/2
               x2: 20 y2: 90 angle2: 0 pull2: 1/2
               contents:
               (list (τ Add-Curve
                        x1: 20 y1: 10 angle1: 180 pull1: 1/2
                        x2: 20 y2: 90 angle2: 180 pull2: 1/2
                        contents:
                        (list (τ Rectangle width: 40 height: 100
                                 outline-mode: "solid" pen-or-color:
"black"))
                        pen-or-color: "white"))
               pen-or-color: "white"))
```

```
> (@ render (τ Scene+Curve
               x1: -20 y1: -20 angle1: 0 pull1: 1
               x2: 120 y2: 120 angle2: 0 pull2: 1
               contents:
               (list (τ Rectangle width: 100 height: 100
```

```
                outline-mode: "solid" pen-or-color: "black"))
        pen-or-color: "red"))
```