

# OCTA workshop

## the Order & Complexity Toolbox for Aesthetics

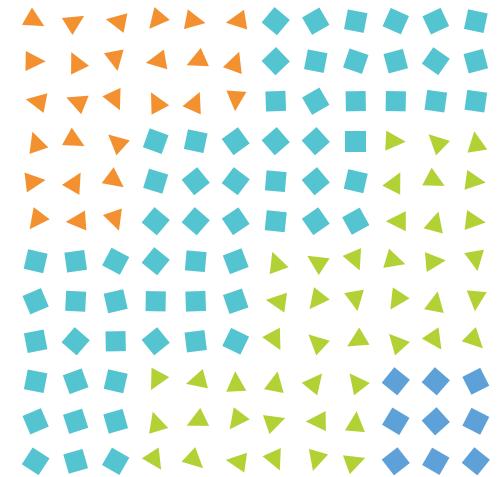
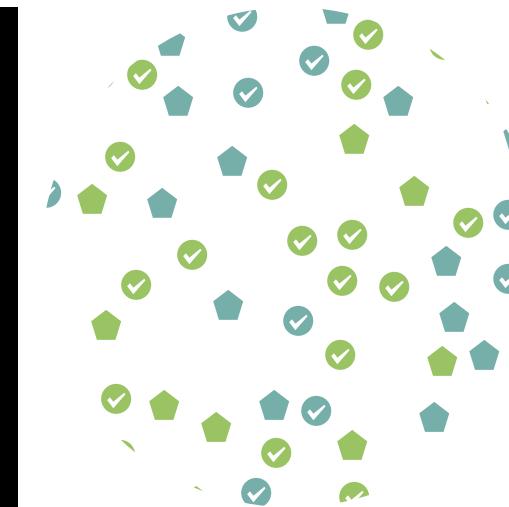
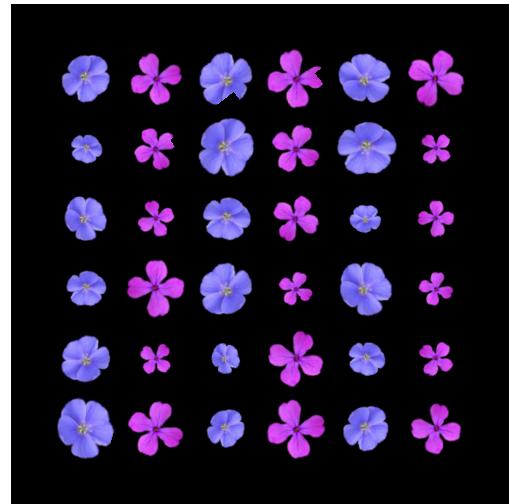
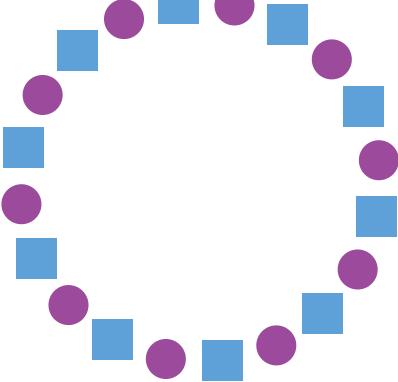
Eline Van Geert

KU Leuven, Belgium

May 9th, 2024

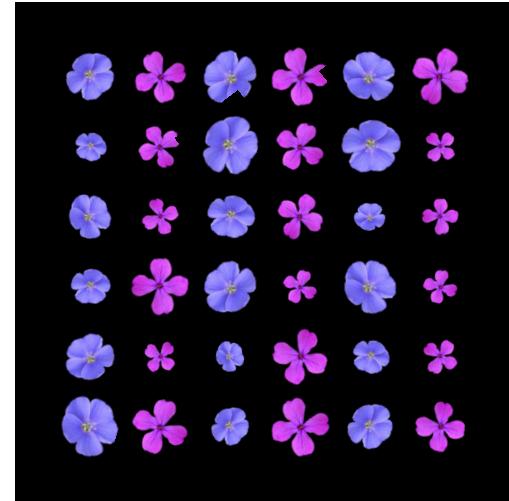


# OCTA workshop: What will you learn today?



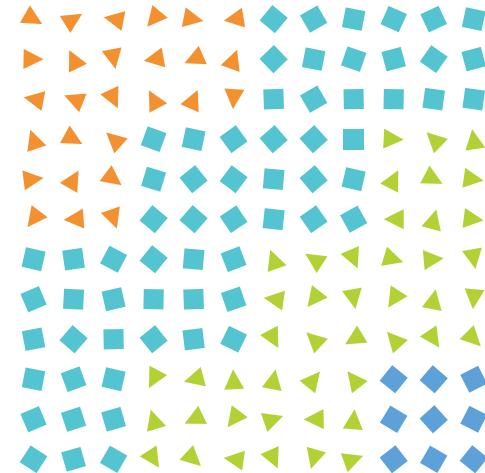
# What will you learn today? Part I

- What is OCTA and why was it created?
- OCTA BASICS (app / code):
  - Installation and prerequisites
  - How to create an OCTA stimulus?
  - Stimulus types & features
  - Positions
  - Element features & feature patterns
  - Deviations
  - Order & complexity
  - Output options



# What will you learn today? Part II

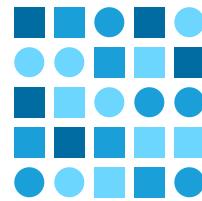
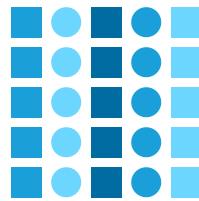
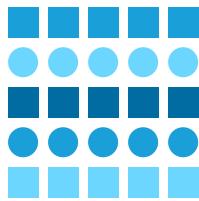
- OCTA ADVANCED (code):
  - Dynamic features
  - Generating stimulus sets
  - Combining OCTA displays
- Use OCTA in your research (app / code)



# OCTA: What and why?

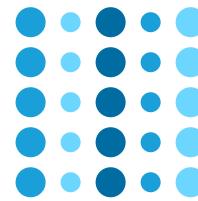
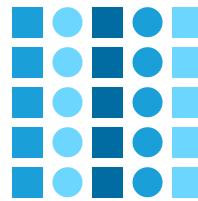
# Order and complexity?

## Order



aspects related to the structure and organization of information in a stimulus

## Complexity

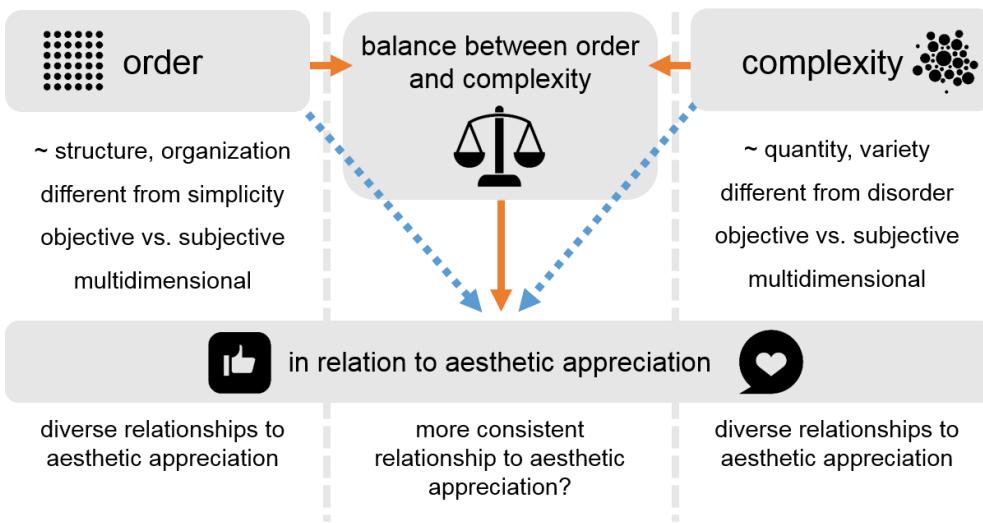


aspects related to the quantity and variety of information in a stimulus

Van Geert, E., & Wagemans, J. (2020). Order, Complexity, and Aesthetic Appreciation. *Psychology of Aesthetics, Creativity, and the Arts*.

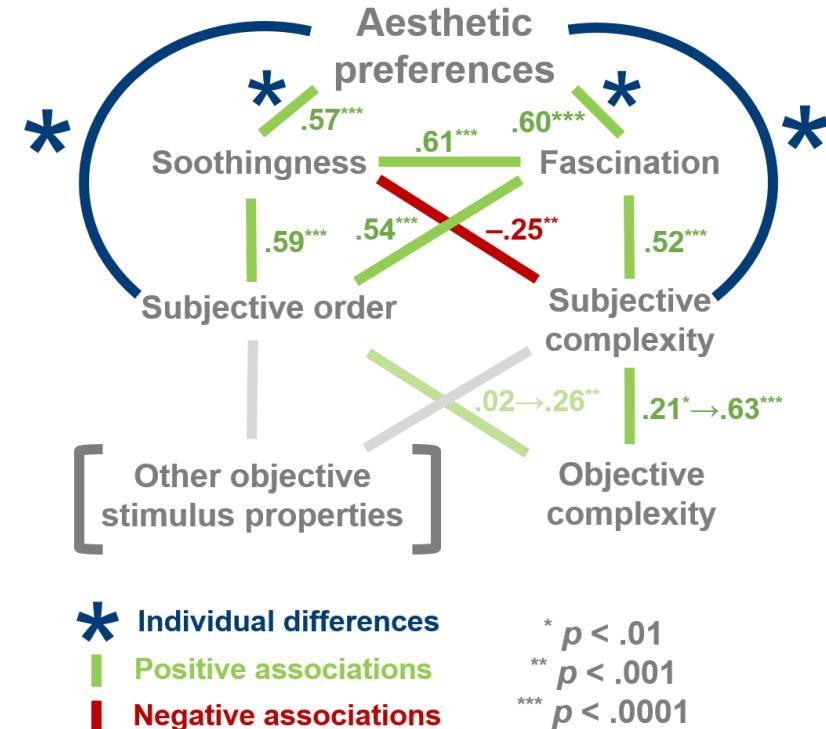
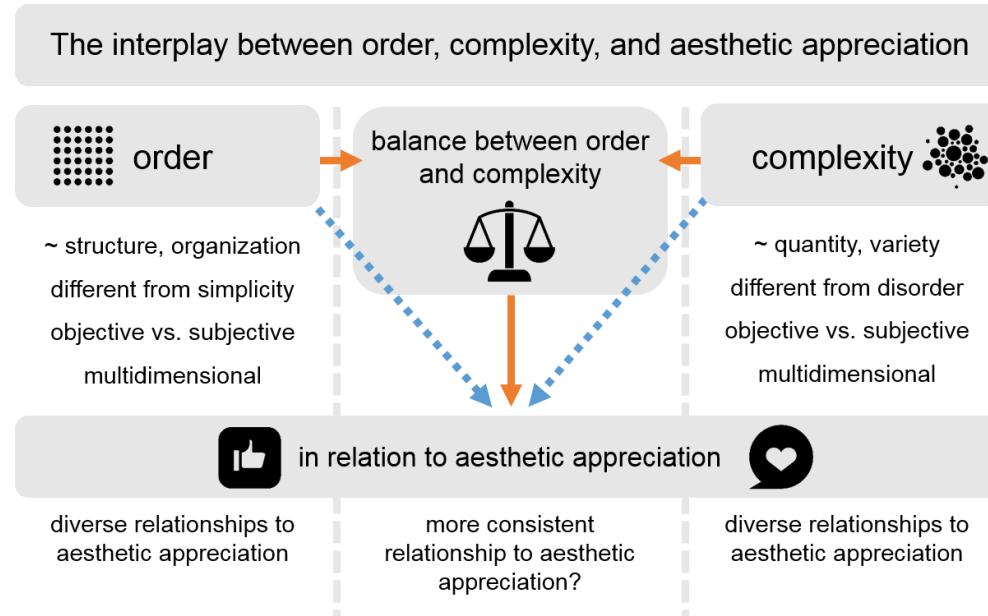
# Order, complexity, and aesthetic appreciation

The interplay between order, complexity, and aesthetic appreciation



Van Geert, E., & Wagemans, J. (2020). Order, Complexity, and Aesthetic Appreciation. *Psychology of Aesthetics, Creativity, and the Arts*.

# Order, complexity, and aesthetic appreciation



Van Geert, E., & Wagemans, J. (2020). Order, Complexity, and Aesthetic Appreciation. *Psychology of Aesthetics, Creativity, and the Arts*.

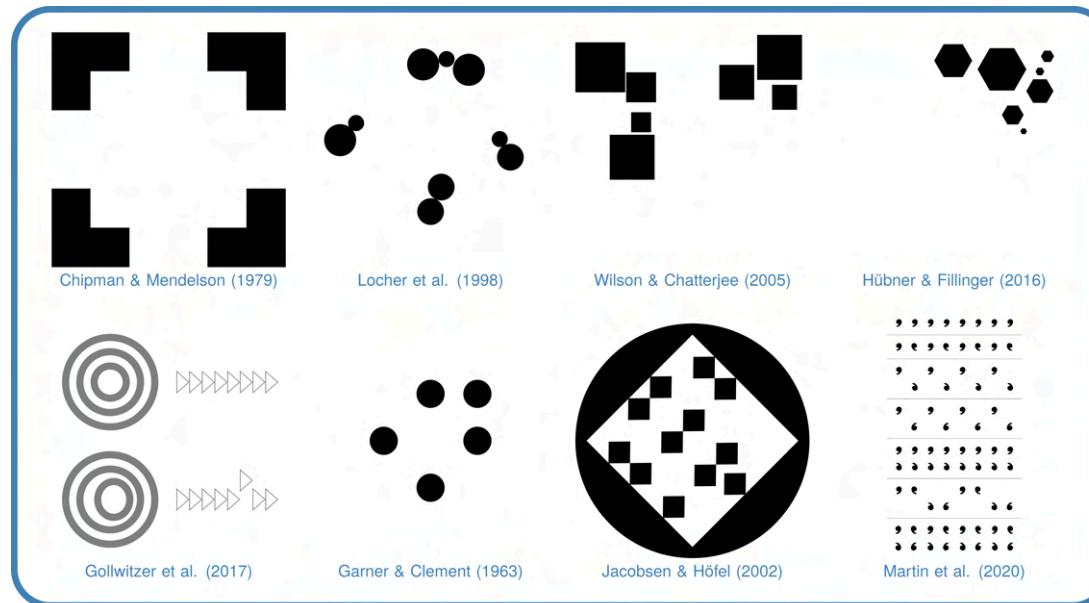
Van Geert, E., & Wagemans, J. (2019). Order, Complexity, and Aesthetic Preferences for Neatly Organized Compositions. *Psychology of Aesthetics, Creativity, and the Arts*.

# Earlier research

- effects of order and complexity often studied in isolation

# Earlier research

- effects of order and complexity often studied in isolation
- when studied jointly, no parametric manipulations or a narrow, unidimensional view



Van Geert, E., Bossens, C., & Wagemans, J. (2023). The Order & Complexity Toolbox for Aesthetics (OCTA): A systematic approach to study the relations between order, complexity, and aesthetic appreciation. *Behavior Research Methods*, 55, 2423–2446. <https://doi.org/10.3758/s13428-022-01900-w>

# Earlier research

- effects of order and complexity often studied in isolation
- when studied jointly, no parametric manipulations or a narrow, unidimensional view



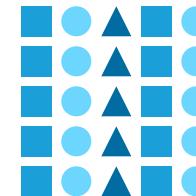
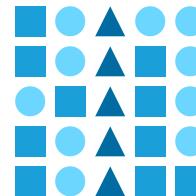
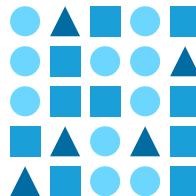
Van Geert, E., & Wagemans, J. (2019). Order, Complexity, and Aesthetic Preferences for Neatly Organized Compositions. *Psychology of Aesthetics, Creativity, and the Arts*.

# Creating OCTA

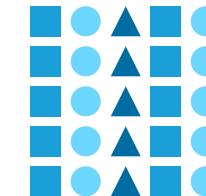
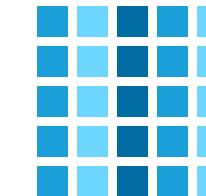


- the Order and Complexity Toolbox for Aesthetics (OCTA)
- create an easy way to generate stimuli varying in complexity and/or order along multiple stimulus dimensions (e.g., shape, color, size, orientation)

## Order



## Complexity

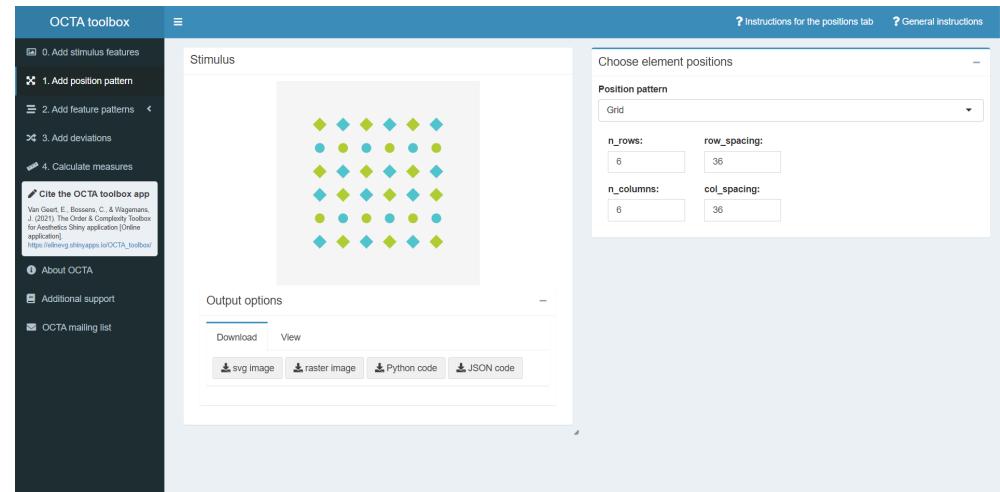


Van Geert, E., Bossens, C., & Wagemans, J. (2023). The Order & Complexity Toolbox for Aesthetics (OCTA): A systematic approach to study the relations between order, complexity, and aesthetic appreciation. *Behavior Research Methods*, 55, 2423–2446. <https://doi.org/10.3758/s13428-022-01900-w>

# Creating OCTA

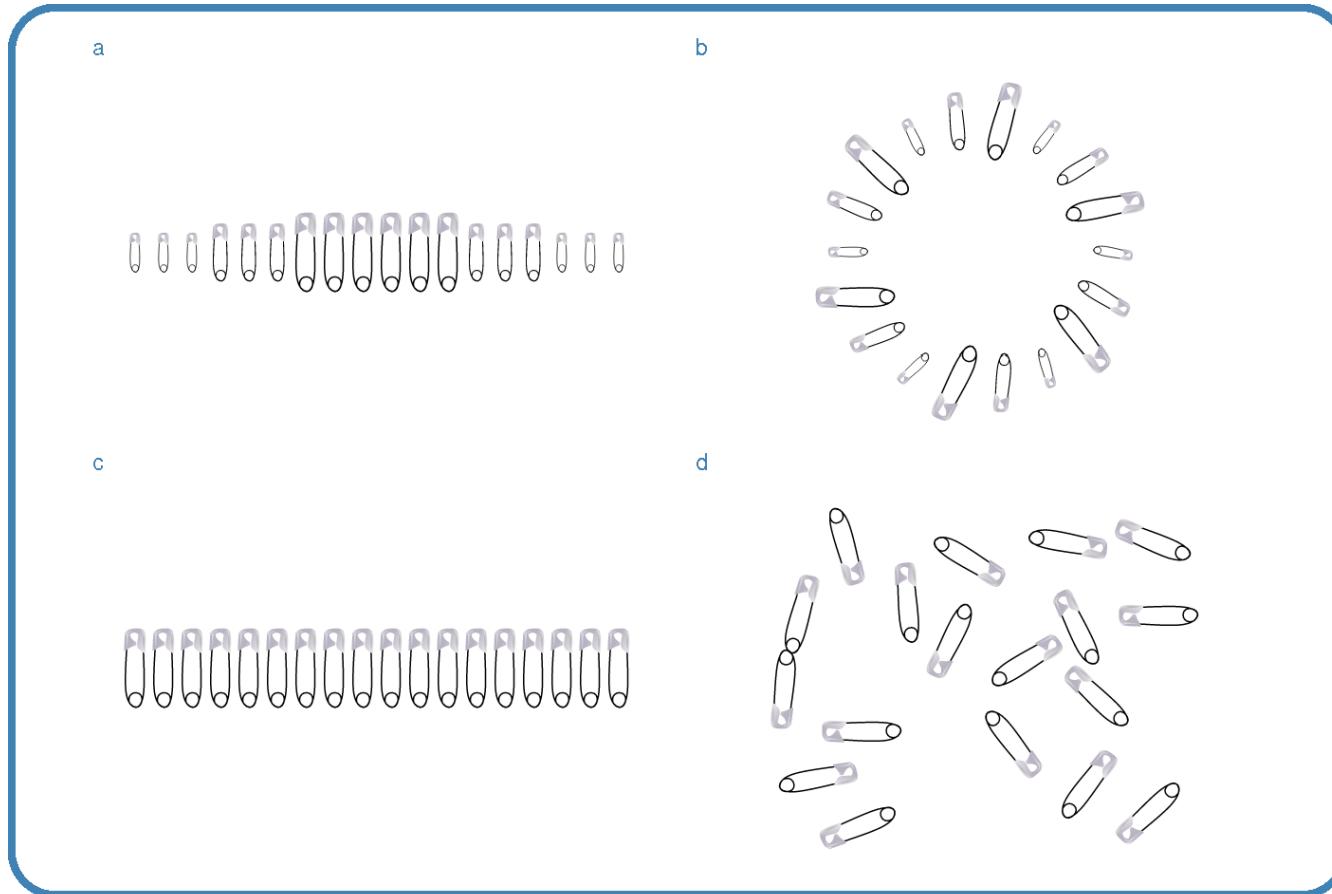
openly available  
reproducible stimuli  
aesthetic appreciation  
**complexity**  
free order  
expandible stimulus sets  
systematic variations  
Shiny application  
Python package

```
1  """
2  OCTA toolbox: demo
3  """
4
5  from octa.Stimulus import Grid
6  from octa.patterns import GridPattern
7  from octa.shapes import Ellipse, Rectangle, Triangle
8
9  ## Create new stimulus
10 stim = Grid(n_rows = 6, n_cols = 6, background_color = "none",
11             row_spacing = 40, col_spacing = 40)
12
13 ## Determine shape of elements used in the stimulus
14 stim.shapes = GridPattern.RepeatAcrossColumns([Rectangle, Triangle, Ellipse])
15
16 ## Determine color of elements used in the stimulus
17 colors_to_use = ["#1b9fd8", "#6dd6ff", "#006ca1"]
18 stim.fillcolors = GridPattern.RepeatAcrossColumns(colors_to_use)
19
20 ## Determine size of elements used in the stimulus
21 stim.boundingboxes = GridPattern.RepeatAcrossColumns([(30,30)])
22
23 stim.Show()
```

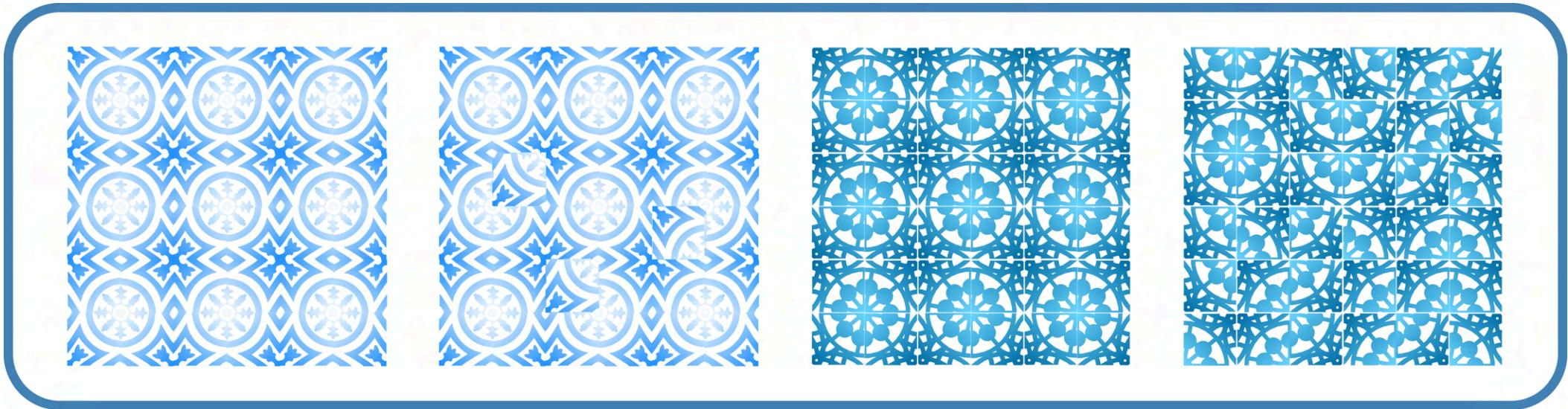


The screenshot shows the OCTA toolbox shiny application. On the left, a sidebar menu lists: 0. Add stimulus features, 1. Add position pattern (selected), 2. Add feature patterns, 3. Add deviations, 4. Calculate measures. Below this are links to 'Cite the OCTA toolbox app' (Van Geert E., Bosscher, C., & Wagenaar, J. (2021). The Order & Complexity Toolbox for Raster Stimulus Application [Online application]. [https://ellenvg.shinyapps.io/OCTA\\_toolbox](https://ellenvg.shinyapps.io/OCTA_toolbox)), 'About OCTA', 'Additional support', and 'OCTA mailing list'. The main area is titled 'Stimulus' and displays a 6x6 grid of colored shapes (triangles, circles, diamonds) in a grid pattern. To the right, a panel titled 'Choose element positions' contains a 'Position pattern' dropdown set to 'Grid', and input fields for 'n\_rows' (6) and 'row\_spacing' (36), and 'n\_columns' (6) and 'col\_spacing' (36). At the bottom, 'Output options' include 'Download' and 'View' buttons, and buttons for 'svg image', 'raster image', 'Python code', and 'JSON code'.

# Going beyond geometric shapes



More flexible, open, and reproducible alternative  
for existing closed tools (e.g., Flextiles)



# Benefits of OCTA

## Benefits of OCTA

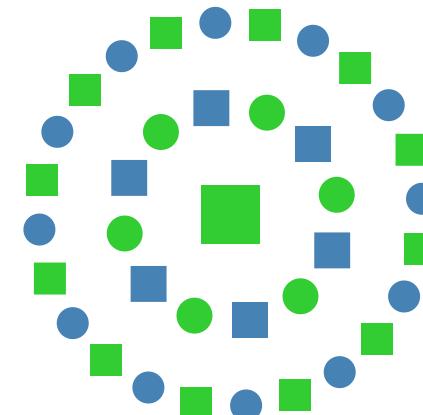
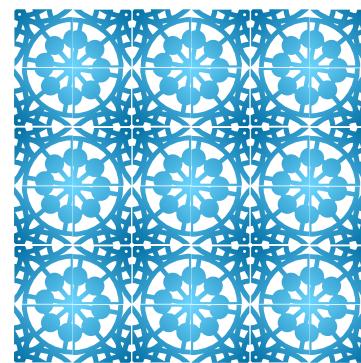
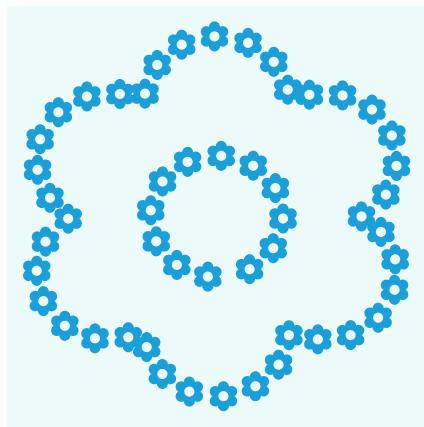
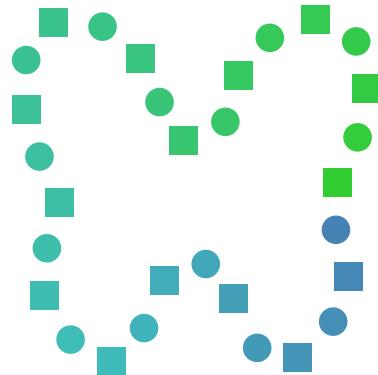
### Previous research

-  often **investigated** order and complexity **separately**
-  often **neglected** the **multidimensionality** of order and complexity and focused on specific aspects of order and complexity
-  often **lacked parametric control** or focused on binary classification when studying order and complexity in combination
-  when parametrically controlled, often **focused on** black-and-white stimuli containing **geometric shapes**, providing only raster-based images as output
-  often **used stimuli created** in proprietary software and/or in a **non-reproducible way**, and/or did not share a detailed stimulus generation procedure
-  usually did **not** provide an easy, accessible way to **recreate or adapt** stimuli to research with or without **programming experience**

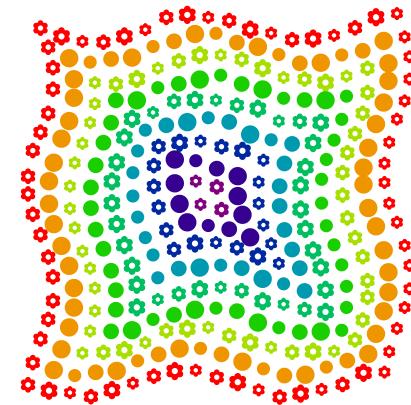
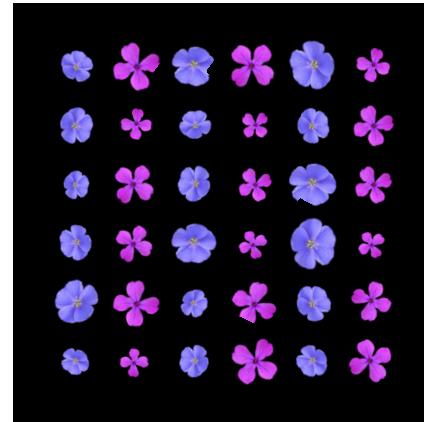
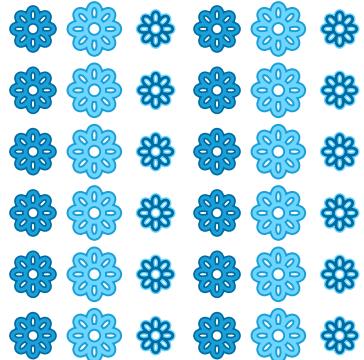
### OCTA

-  allows for **studying** order and complexity **in combination**
-  allows for diverse order and complexity **manipulations on multiple feature dimensions**
-  allows for **parametrically controlled** order and complexity manipulations, going beyond binary classification
-  allows for **ecological validity** and the study of **dynamics over time** by the option to add **more complex path or image elements and animated feature values**, and provides more diverse output options
-  provides a **free, open source** tool that produces easily **reproducible** or adaptable stimulus **output**
-  allows both researchers with and without programming experience to **easily reproduce or adapt** existing **stimuli**, as both a Python package and an online user interface are provided

# Example OCTA stimuli

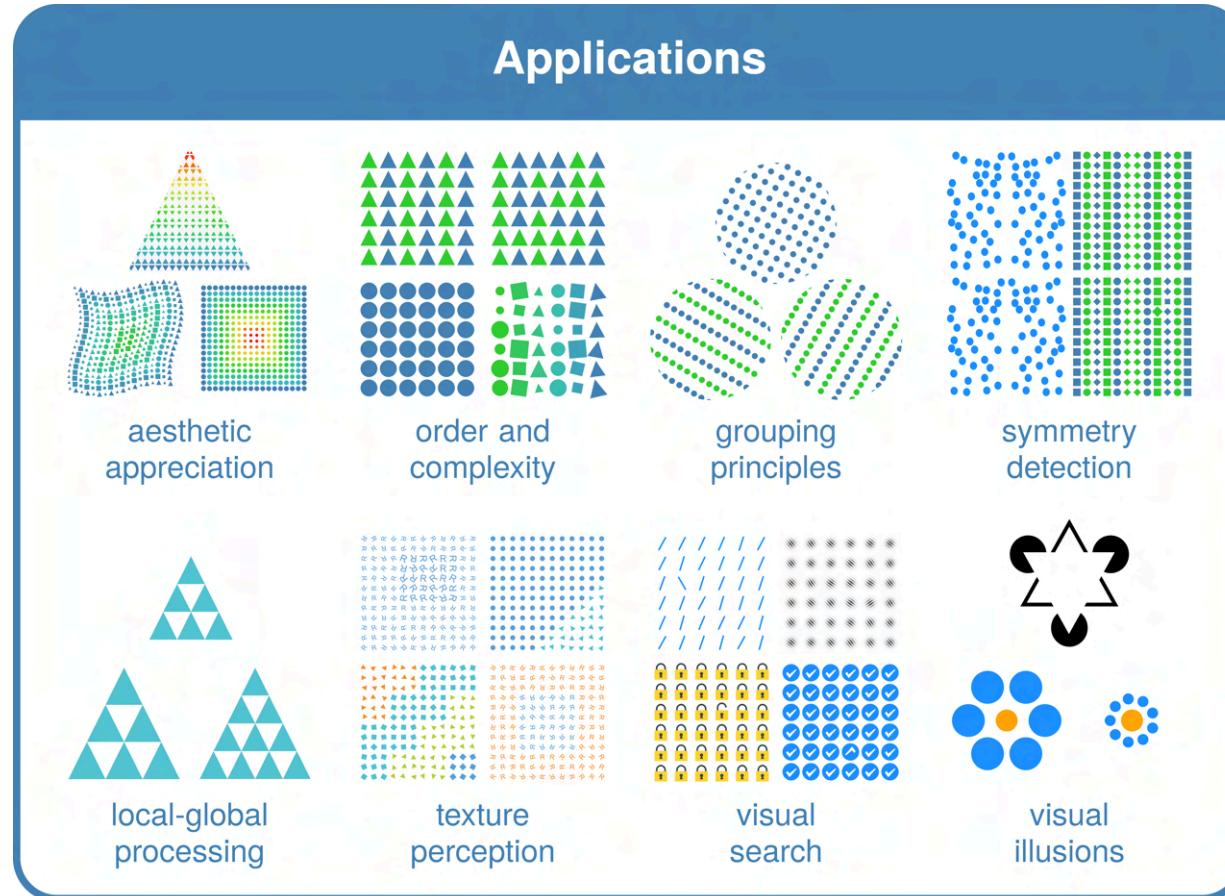


Recreated based on [the official IAEA 2021 logo](#)



Flower images from Hůla and Flegr (2016)

# Applications



# Cite OCTA when using it!

Always cite OCTA when using it in your work;

Van Geert, E., Bossens, C., & Wagemans, J. (2023). The Order & Complexity Toolbox for Aesthetics (OCTA): A systematic approach to study the relations between order, complexity, and aesthetic appreciation. *Behavior Research Methods*, 55, 2423–2446. <https://doi.org/10.3758/s13428-022-01900-w>

If you also want to refer to the Python toolbox or the Shiny app specifically, you can use:

- for the Python toolbox: Van Geert, E., Bossens, C., & Wagemans, J. (2021). The Order & Complexity Toolbox for Aesthetics Python library [Computer software]. [https://github.com/gestaltrevision/OCTA\\_toolbox](https://github.com/gestaltrevision/OCTA_toolbox)
- for the app: Van Geert, E., Bossens, C., & Wagemans, J. (2021). The Order & Complexity Toolbox for Aesthetics Shiny application [Online application]. [https://elinevg.shinyapps.io/OCTA\\_toolbox/](https://elinevg.shinyapps.io/OCTA_toolbox/)

# OCTA: Let's start!

# OCTA: App demo

# Workshop documents

Find the workshop materials and click on the 'Launch binder' icon: [https://github.com/ElineVG/OCTA\\_workshop](https://github.com/ElineVG/OCTA_workshop)

Alternatively, you can access the workshop here:

<https://colab.research.google.com/drive/1v5AVb76tQ-aGVdJFvKi5Rqrw4luRMztP?usp=sharing>

If you want to work with the app **during** the workshop, also open one of these links:

- [https://octa.shinyapps.io/OCTA\\_app1/](https://octa.shinyapps.io/OCTA_app1/)
- [https://octa.shinyapps.io/OCTA\\_app2/](https://octa.shinyapps.io/OCTA_app2/)
- [https://octa.shinyapps.io/OCTA\\_app3/](https://octa.shinyapps.io/OCTA_app3/)
- [https://octa.shinyapps.io/OCTA\\_app4/](https://octa.shinyapps.io/OCTA_app4/)

**After** the workshop, access the app here:

- [https://elinevg.shinyapps.io/OCTA\\_toolbox/](https://elinevg.shinyapps.io/OCTA_toolbox/)

# Installation and requirements: APP

No installation needed! Open the app in your browser:

If you want to work with the app **during** the workshop, also open one of these links:

- [https://octa.shinyapps.io/OCTA\\_app1/](https://octa.shinyapps.io/OCTA_app1/)
- [https://octa.shinyapps.io/OCTA\\_app2/](https://octa.shinyapps.io/OCTA_app2/)
- [https://octa.shinyapps.io/OCTA\\_app3/](https://octa.shinyapps.io/OCTA_app3/)
- [https://octa.shinyapps.io/OCTA\\_app4/](https://octa.shinyapps.io/OCTA_app4/)

After the workshop, access the app here:

- [https://elinevg.shinyapps.io/OCTA\\_toolbox/](https://elinevg.shinyapps.io/OCTA_toolbox/)

# Installation and requirements: PYTHON

To run octa online:

- surf to [https://github.com/ElineVG/OCTA\\_workshop](https://github.com/ElineVG/OCTA_workshop)
- click on 'launch binder'

(OR alternatively, open this link: <https://colab.research.google.com/drive/1v5AVb76tQ-aGVdJFvKi5Rqrw4IuRMztP?usp=sharing>)

To run octa locally on your computer:

- Open the software you use to open Python documents (Spyder / PyCharm / ...)
- install the following packages (or check whether already installed): `svgwrite` `svg.path` `svgpathtools` `svgutils` `jsonpickle` `html2image` `svglib` `reportlab` `colour` `IPython` `pandas`
- install octa: `pip install octa`

# Installation and requirements: PYTHON

Load the most common functions part of the octa package:

```
from octa.Stimulus import Grid, Outline, Concentric, Stimulus
from octa.Positions import Positions
from octa.patterns import GridPattern, Pattern, Sequence, LinearGradient
from octa.shapes import (Ellipse, Rectangle, Triangle, Polygon, RegularPolygon,
                        Path, PathSvg, Image, FitImage, Text)
from octa.measurements import Order, Complexity
```

# How to create an OCTA stimulus?

How to create a stimulus with OCTA?

0. Import octa

```
from octa.Stimulus import Grid, Outline, Concentric
from octa.Positions import Positions
from octa.patterns import GridPattern
from octa.shapes import Ellipse, Rectangle, Triangle
from octa.measurements import Order, Complexity
```
1. Specify stimulus type

```
stim = Grid(n_rows = 9, n_cols = 9)
```
- [2. Customize positions]
- 3a. Specify boundingboxes

```
stim.boundingboxes = GridPattern.RepeatAcrossElements( [ (45,45), (30,30) ] )
```
- 3b. Specify shapes

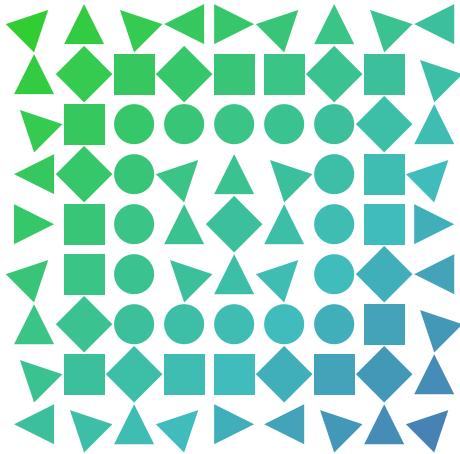
```
stim.shapes = GridPattern.RepeatAcrossLayers( [ Rectangle, Triangle, Ellipse ] )
```
- 3c. Specify fillcolors

```
stim.fillcolors = GridPattern.GradientAcrossRightDiagonal( start_value = 'limegreen', end_value = 'steelblue' )
```
- 3d. Specify orientations

```
stim.orientations = GridPattern.MirrorAcrossLeftDiagonal( [ -90, -45, 0, 45, 90 ] )
```
- [3e. Add additional features]
- [4. Add deviations]
5. Show and save stimulus

```
stim.Show()
stim.SaveSVG("SVGfilename", folder = "foldername")
stim.SaveJSON("JSONfilename", folder = "foldername")
stim.SavePNG("PNGfilename", scale = 10, folder = "foldername")
```

# Create your first OCTA stimulus!



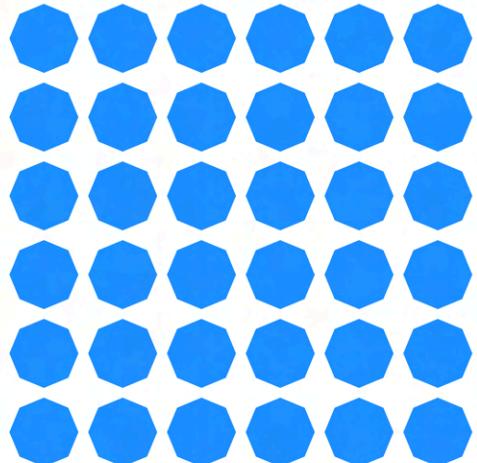
```
stimulus = Grid(9,9, row_spacing=25, col_spacing=25)
stimulus.boundingboxes = GridPattern.RepeatAcrossElements([(20,20)])
stimulus.shapes = GridPattern.RepeatAcrossLayers([Rectangle, Triangle, Ellipse])
stimulus.fillcolors = GridPattern.GradientAcrossRightDiagonal(start_value = 'limegreen', end_val
stimulus.orientations = GridPattern.MirrorAcrossLeftDiagonal([-90,-45,0,45,90])
stimulus.Show()
```

# Exercise 1

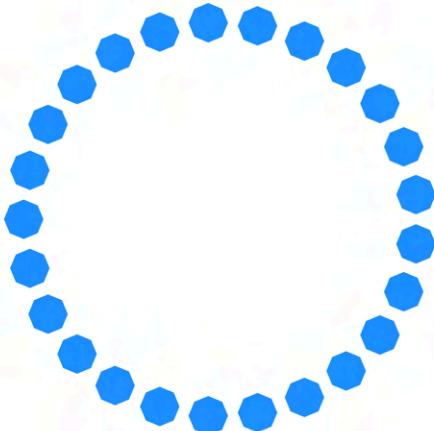
1. Click the 'Run Code' button to create your first OCTA stimulus.
2. Can you change the number of rows and columns to 12 instead of 9?
3. Can you change the elements' colors to go from red to limegreen?
4. Can you change the color gradient to go across the rows (`AcrossRows`) instead of across the right diagonal (`AcrossRightDiagonal`)?

# 1. Specify stimulus type

## Stimulus types



Grid

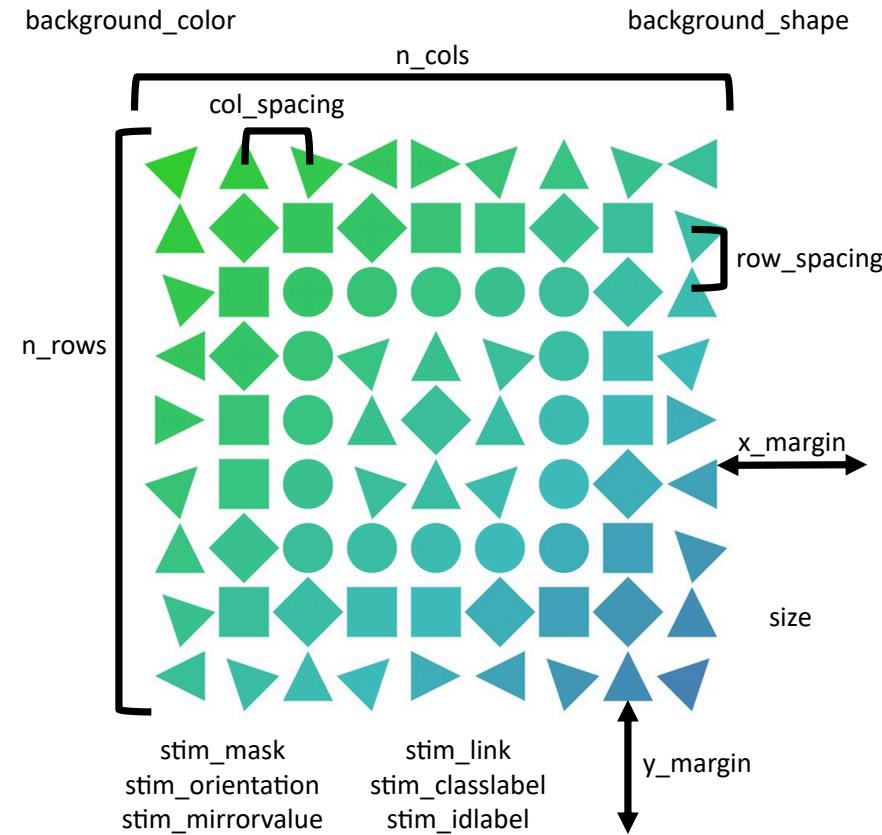


Outline



Concentric

# 1. Specify stimulus characteristics



# 1. Specify stimulus characteristics

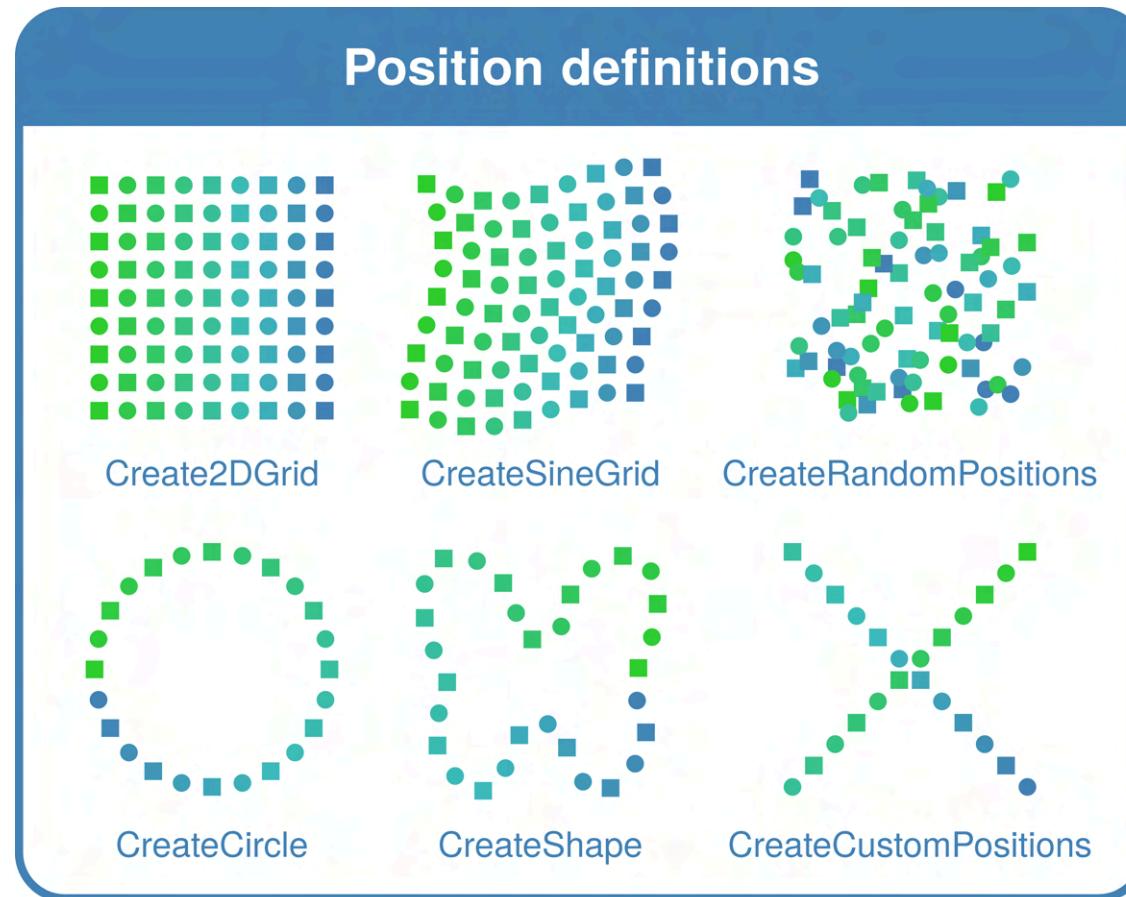
When defining stimulus type:

```
stimulus = Grid(n_rows = 3, n_cols = 6, background_color = "gainsboro")
stimulus.Show()
```

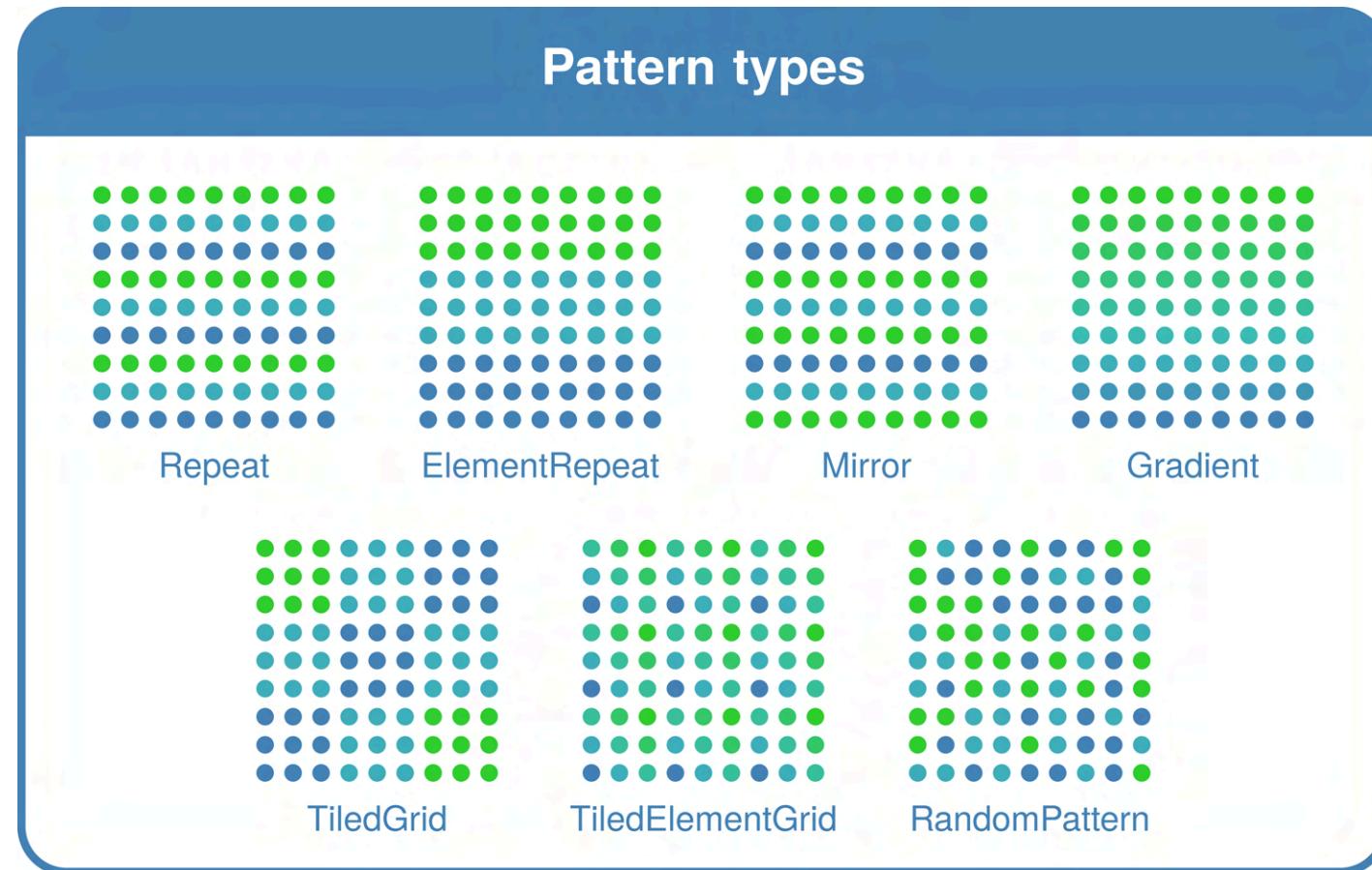
Or after stimulus has been created:

```
stimulus = Grid(3,3)
stimulus.n_cols = 6
stimulus.background_color = "gainsboro"
stimulus.Show()
```

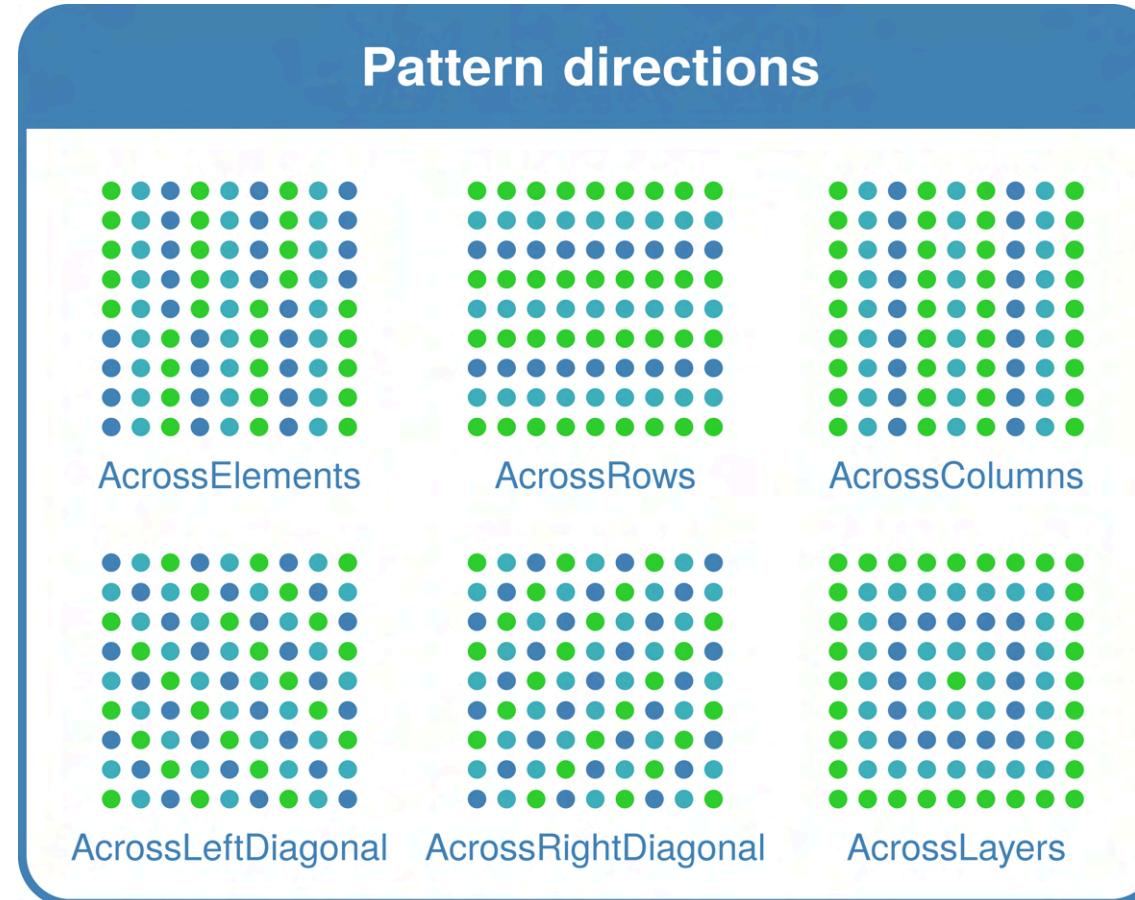
## 2. Customize positions



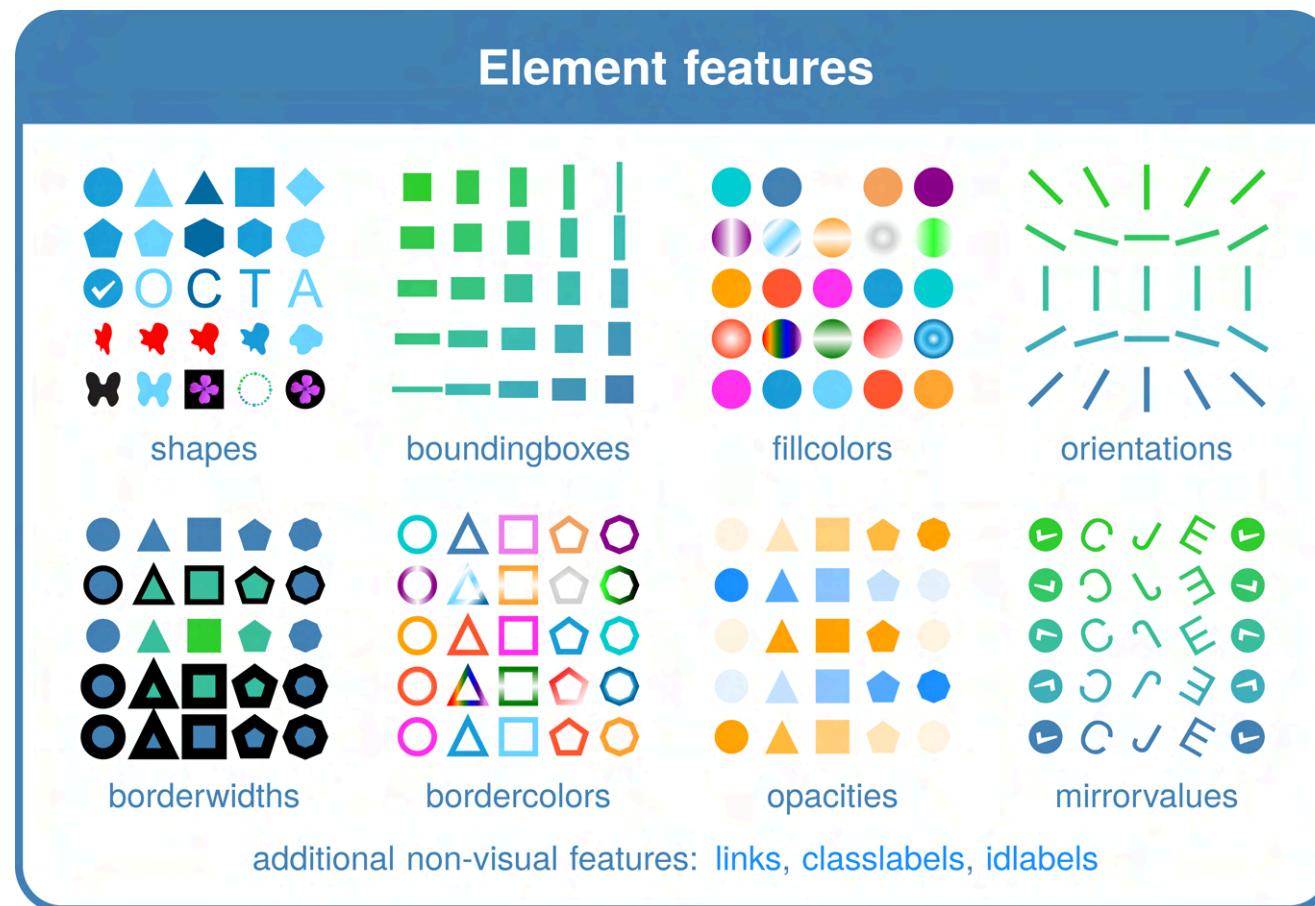
### 3. Specify pattern types



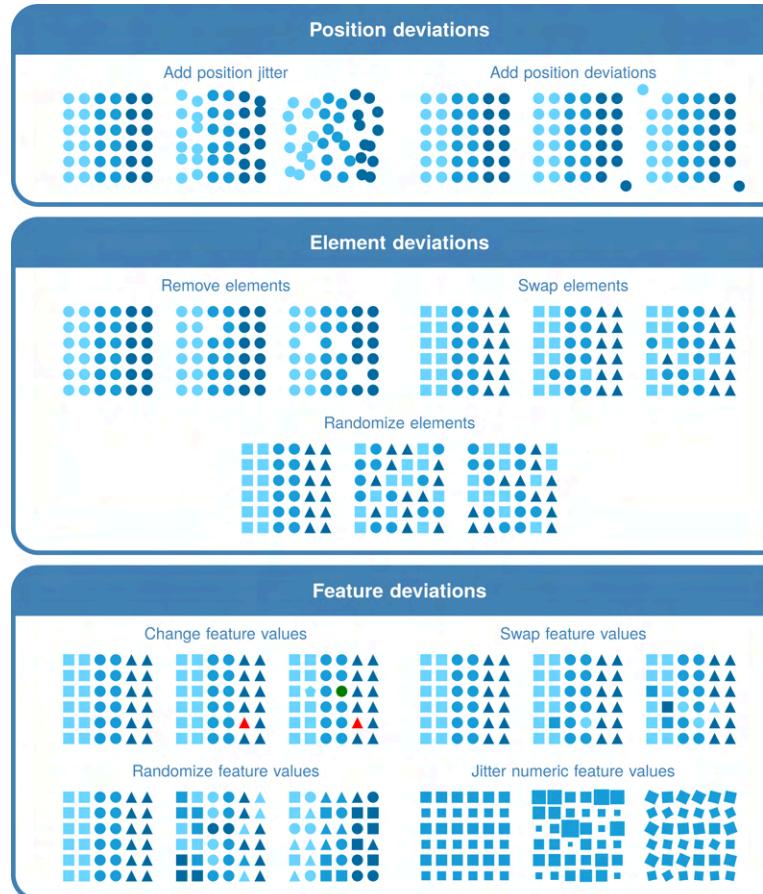
### 3. Specify pattern directions



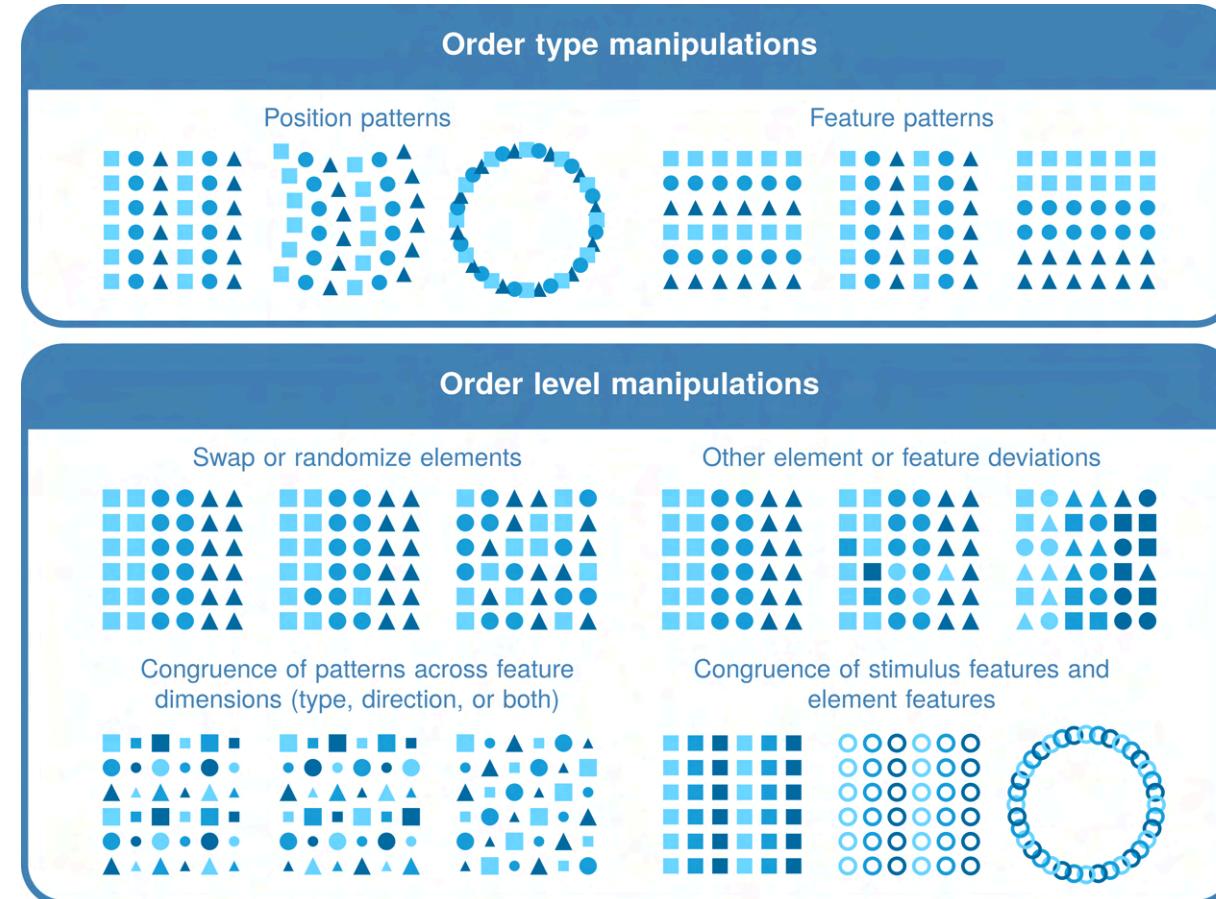
# 3. Specify element features



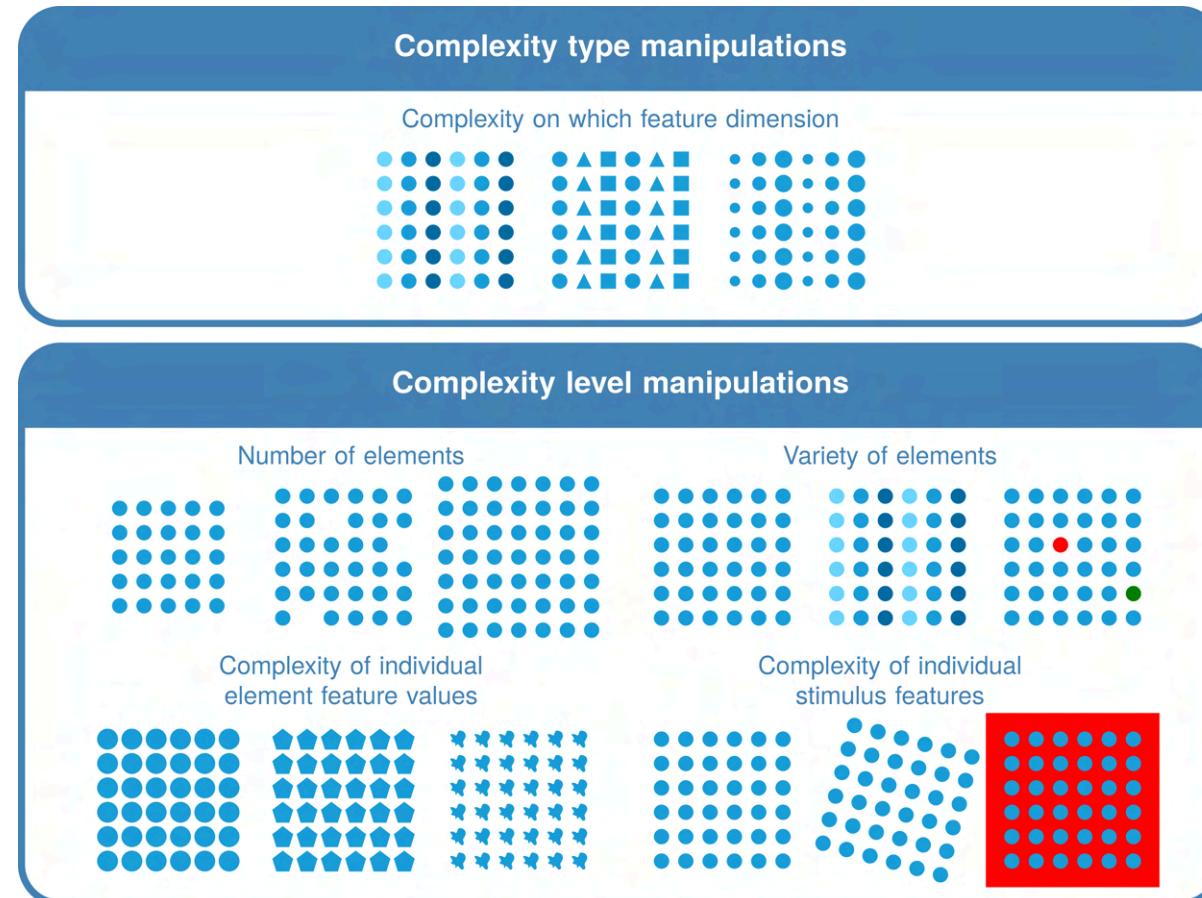
# 4. Add deviations



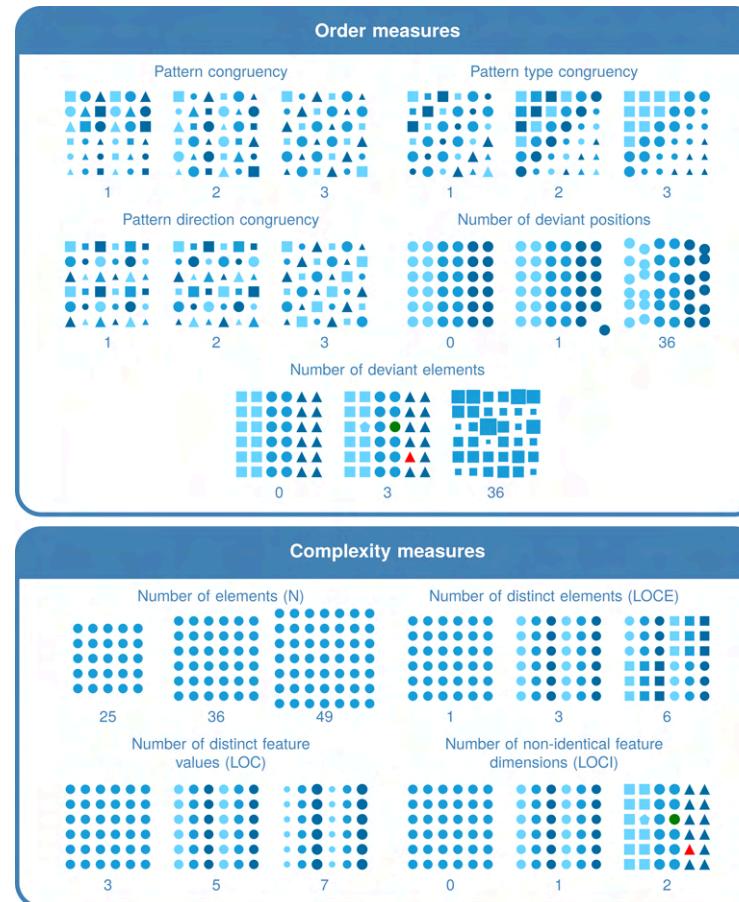
# 4. Order manipulations



# 4. Complexity manipulations



# 4. Order & complexity measures



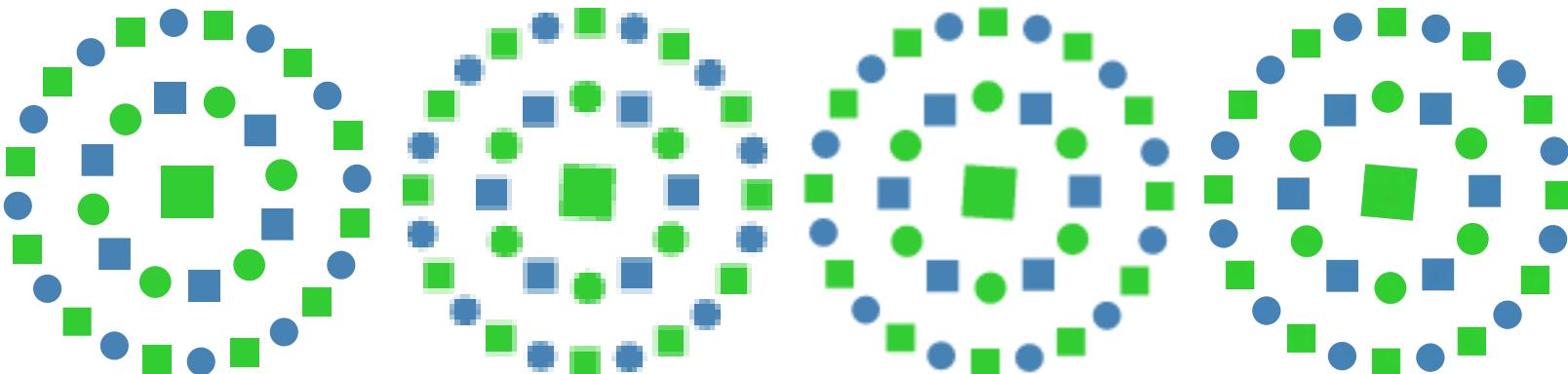
# 5. Show & save stimulus

- stimulus as vector image (SVG)
- stimulus as raster image (PNG, JPG, PDF, TIFF)
- computer-readable JSON output to recreate the stimulus
- (in Python with the LoadFromJSON function)

In the online Shiny app: download or view the Python code to (re)create the stimulus in Python

Set a seed to reproduce the exact same random deviations!

Scale option to increase quality for raster images [SVG - PNG scale 0.5 - PNG scale 1 - PNG scale 10]:



# 5. Show and save stimulus

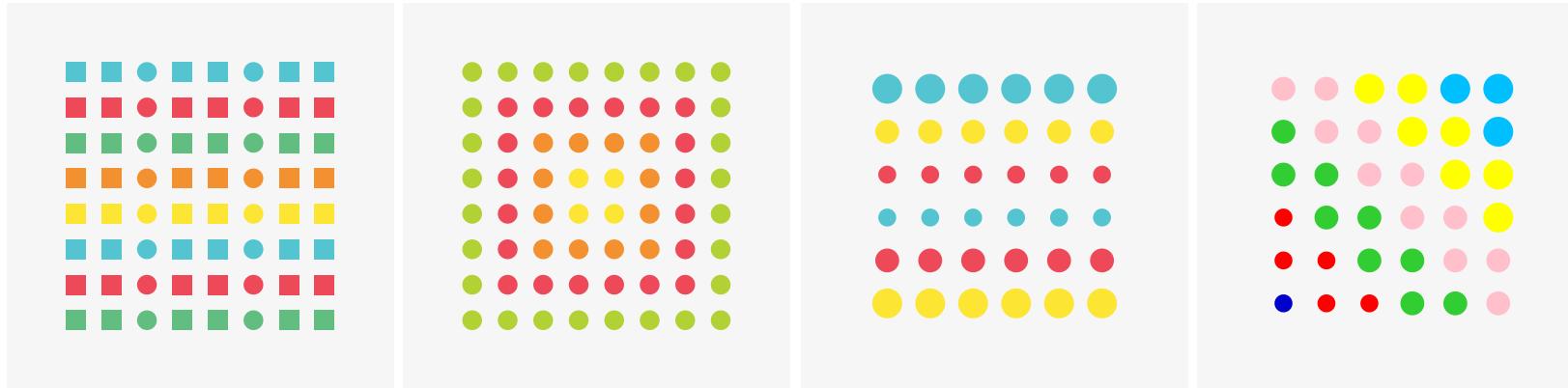
Points of attention:

- Always add random.seed(..) just before each SaveSVG/PNG/JPG/... function (because the stimulus is regenerated for each of those). Otherwise, random values will differ between different output files.
- Always stimulus.Render() when using GetSVG(). Otherwise, the stimulus will not be rendered automatically and values will not be updated.
- Be aware that **svg** may be removed when you add **SavePNG** or **SaveJPG**! Therefore it is best to save each output type in a separate folder.

# Terminology

Terminology used in OCTA	
order	aspects related to the structure and organization of elements in a stimulus
complexity	aspects related to the quantity and variety of elements in a stimulus
stimulus	overall display, total configuration including one or more elements
element	object, part of stimulus display which is usually repeated and spatially separated from other parts
stimulus feature	certain feature of the stimulus as a whole (e.g., size, orientation, background color, background size)
element feature	certain feature of the elements in the stimulus (e.g., shape, size, color, orientation)
(element) position pattern	overall configuration of the element positions in the stimulus (e.g., rectangular grid, sinegrid, circle, shape)
(element) feature pattern	overall configuration of the element features in the stimulus (combination of pattern values, pattern type, and pattern direction)
pattern values	the values of an element feature that will be used in the stimulus
pattern type	type of structure or organization present for a certain feature across the elements in the stimulus (e.g., pattern repeat, element repeat, mirror symmetry, gradient)
pattern direction	the direction in which the pattern values are applied according to the pattern type across the stimulus (e.g., across elements, across rows, across columns)
position deviation	deviation from the position pattern, either by the addition of random jitter to the element positions or by the adaptation or removal of specific element positions
element deviation	deviation from the element feature patterns present in the stimulus, by removing elements from the stimulus, swapping element positions, or randomizing element positions
feature deviation	deviation from at least one element feature pattern, by removing elements from the stimulus, swapping one or more features between elements in the stimulus, randomizing element or feature positions, changing the feature value of one or more random or specified elements in the stimulus, or by jittering one or more numeric element features

# Challenges!



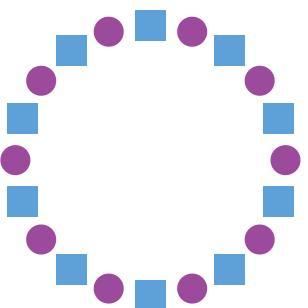
Need some help? Look here: <https://colab.research.google.com/drive/1m5bKqK6oE0qf2J1AUn4CAcvG7Ged698-?usp=sharing>

# Additional challenges!

- Remove the background color (TAB 0)
- Try to use the animated orientation (TAB 0)
- Try different position patterns (e.g., SineGrid; TAB 1)
- Add additional features like border widths or opacities (TAB 2)
- Remove 3 random elements from the display (TAB 3)
- Add uniform position jitter (TAB 3)
- Give 1 element a custom fillcolor (e.g., 'violet', 'black'; TAB 3)
- Find the level of element complexity (LOCE; TAB 4)
- Explore the example stimuli (SUPPORT TAB)

# OCTA: Advanced topics

# Dynamic features

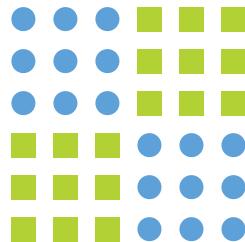


```
stim = outline(n_elements = 20, x_margin = 0, y_margin = 0, shape_boundingbox = (90,90), background_color = "white")
stim.orientation = [ 'animate', '0', '360', "dur='40s'", repeatCount='indefinite' ]
stim.boundingboxes = GridPattern.RepeatAcrossElements( [ (20,20) ] )
stim.shapes = GridPattern.RepeatAcrossElements( [ Ellipse, Rectangle ] )
stim.fillcolors = GridPattern.RepeatAcrossElements( [ "#9C4B9C", "#5EA1D8" ] )

stim.orientations = GridPattern.RepeatAcrossElements(
    [ [ 'animate', '360', '0', "dur='40s'", repeatCount='indefinite' ] ] )

stim.Show()
stim.SaveSVG("stim_example1", folder = "")
```

# Dynamic features



```
stim = Grid(n_rows = 6, n_cols = 6, row_spacing = 35, col_spacing= 35, size = (250,250))
stim.boundingboxes = GridPattern.RepeatAcrossElements( [ (20,20) ] )

stim.fillcolors = GridPattern.TiledElementGrid(source_grid = GridPattern.RepeatAcrossRightDiagon
    [ ['animate', '#5EA1D8', 'values = "#5EA1D8;#B2D135", begin = "0s", calcMode = "discrete",
        dur = "5s", repeatCount = "indefinite"] ,
        ['animate', '#B2D135', 'values = "#B2D135;#5EA1D8", begin = "0s", calcMode = "discrete",
        dur = "5s", repeatCount = "indefinite"] ], 2, 2), tile_multiplier = 3)

stim.shapes = GridPattern.TiledElementGrid(source_grid = GridPattern.RepeatAcrossRightDiagonal(
stim.Show()
stim.SaveSVG("stim_example2", folder = "")
```

# Dynamic features

- animate
- set

More examples in the app under 'Custom' colors, orientations, etc.

More info on SVG animation: [https://www.w3schools.com/graphics/svg\\_animation.asp](https://www.w3schools.com/graphics/svg_animation.asp)

# Dynamic features

No dynamic feature options for shape in OCTA directly, BUT:

- you can add a gif video as shape using the image tag
- you can create a dynamic svg in which the shape changes and add it using the image tag
- you can create static OCTA stimuli and concatenate them in a video (e.g., using moviepy)

# Generating a stimulus set

```
# Load necessary objects and functions from OCTA
from octa.Stimulus import Grid
from octa.patterns import GridPattern
from octa.shapes import Ellipse, Rectangle

# Set variable values to use in stimulus set
cols = ["#5EA1D8", "#B2D135", "#F39130", "#ED4959"]
# ["#9C4B9C", "#5EA1D8", "#54C4D0", "#62BD80", "#B2D135", "#FCE533", "#F39130", "#ED4959"]
colors_in_set1 = []
for i in range(len(cols)):
    for j in range(len(cols)):
        # do not add those conditions where both colors are the same:
        if cols[i] != cols[j]:
            colors_in_set1.append([cols[i], cols[j]])
```

# Generating a stimulus set

```
shapes = [ Ellipse, Rectangle, RegularPolygon(5) ]
shapes_in_set1 = []
for i in range(len(shapes)):
    for j in range(len(shapes)):
        # do not add those conditions where both shapes are the same:
        if shapes[i] != shapes[j]:
            shapes_in_set1.append([shapes[i], shapes[j]])
#shapes_in_set1 = [ [Ellipse, Rectangle], [Ellipse, RegularPolygon(3)], [Ellipse, RegularPolygon(5)] ]
```



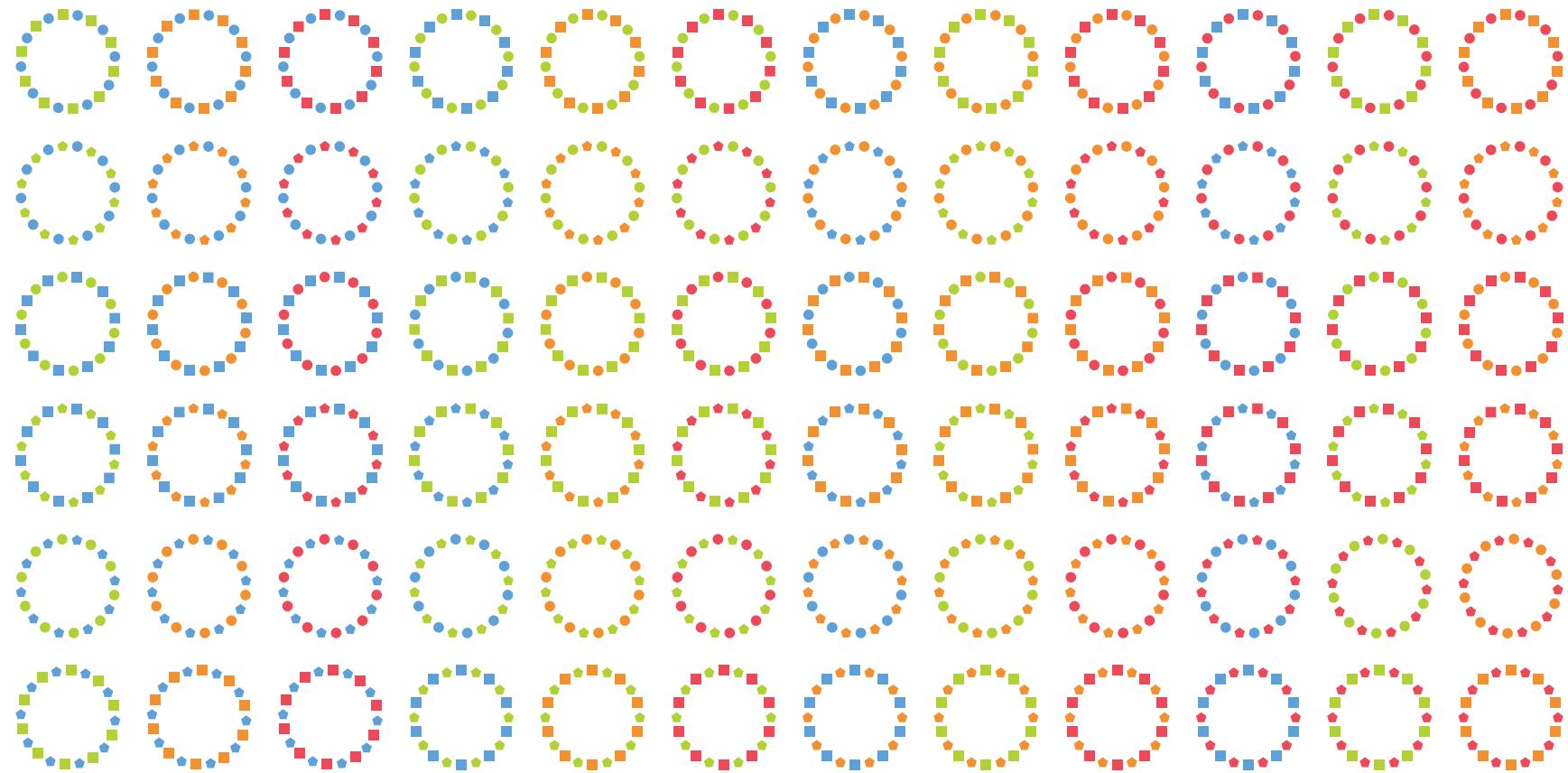
# Generating a stimulus set

```
# Define base stimulus [with features that stay same across all stimuli in set]
stim = Outline(n_elements = 20, x_margin = 0, y_margin = 0, shape_boundingbox = (90,90), backgro
    stim_orientation = ['animate', '0', '360', "dur='40s", repeatCount='indefinit
## Add non-changing features of base stimulus
stim.boundingboxes = GridPattern.RepeatAcrossElements( [(20,20) ] )
stim.orientations = GridPattern.RepeatAcrossElements( [ ['animate', '360', '0', "dur='40s", repe
◀ ━━━━━━ ▶
```

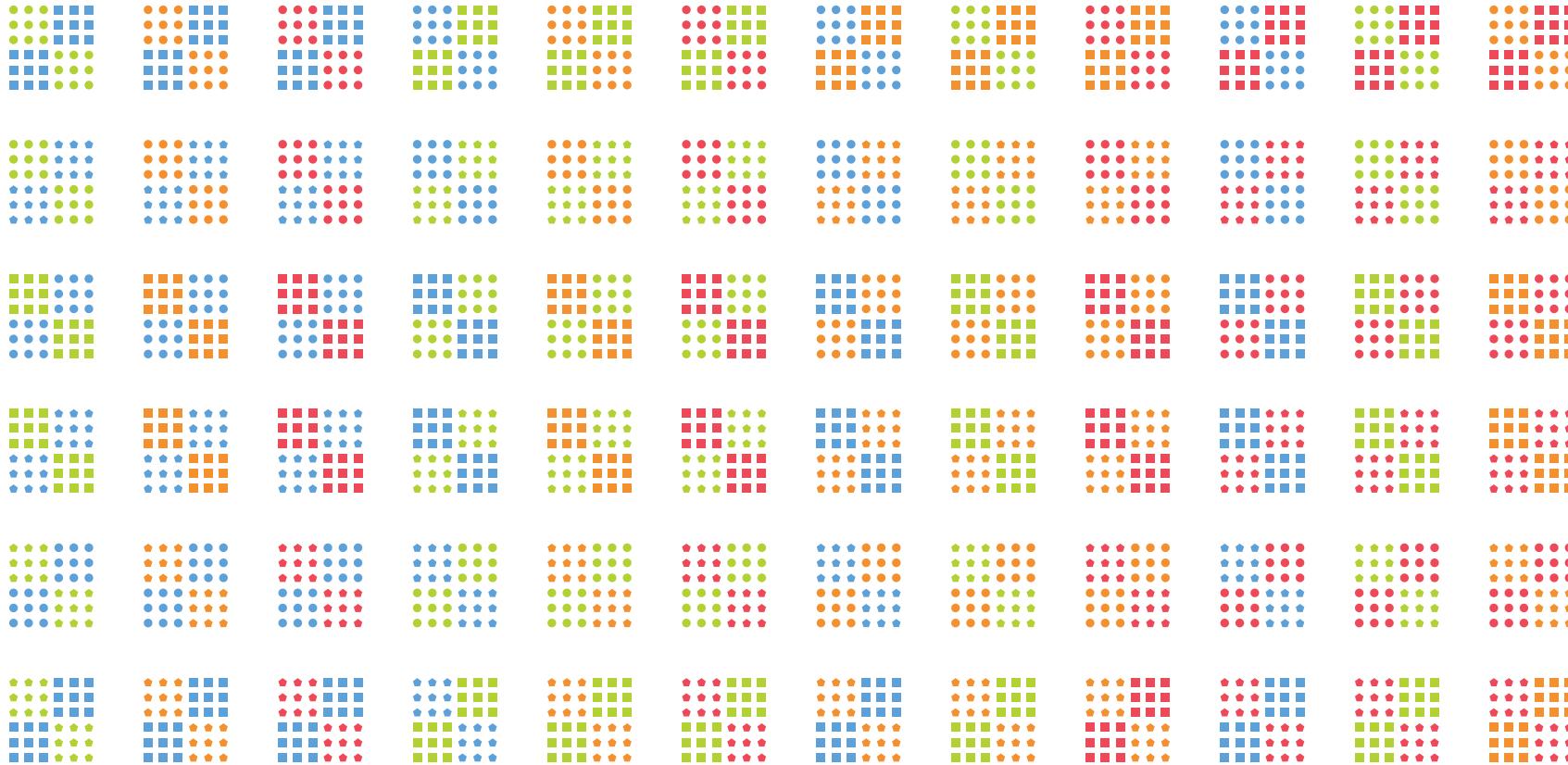
# Generating a stimulus set

```
for chosen_colors in range(len(colors_in_set1)):  
    for chosen_shapes in range(len(shapes_in_set1)):  
  
        # Define stimulus name  
        stimname = "outlinestim_" + str(chosen_shapes+1) + "_" + str(chosen_colors+1)  
  
        # Add fillcolors for the elements  
        stim.fillcolors = GridPattern.RepeatAcrossElements( colors_in_set1[chosen_colors] )  
  
        # Add shapes for the elements  
        stim.shapes = GridPattern.RepeatAcrossElements( shapes_in_set1[chosen_shapes] )  
  
        # Save stimulus  
        stim.Render()  
        #stim.Show()  
        stim.SaveSVG(stimname, folder = "")
```

# Arranging different OCTA displays



# Arranging different OCTA displays



OCTA:  
Use OCTA in your research!

# Use pregenerated OCTA stimuli in your experiments

Please click the image you prefer

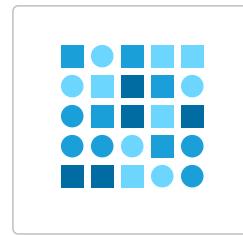
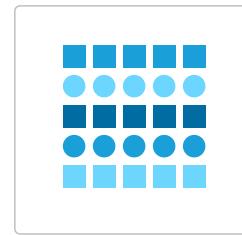
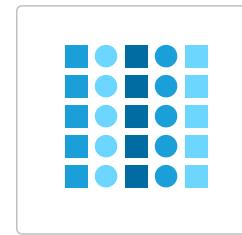
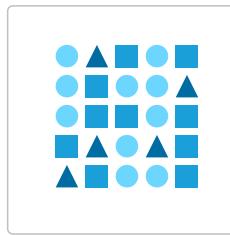
Possible with any software

Psychopy: needs raster images

Demo experiments in jspsych [www.jspsych.org/](http://www.jspsych.org/)

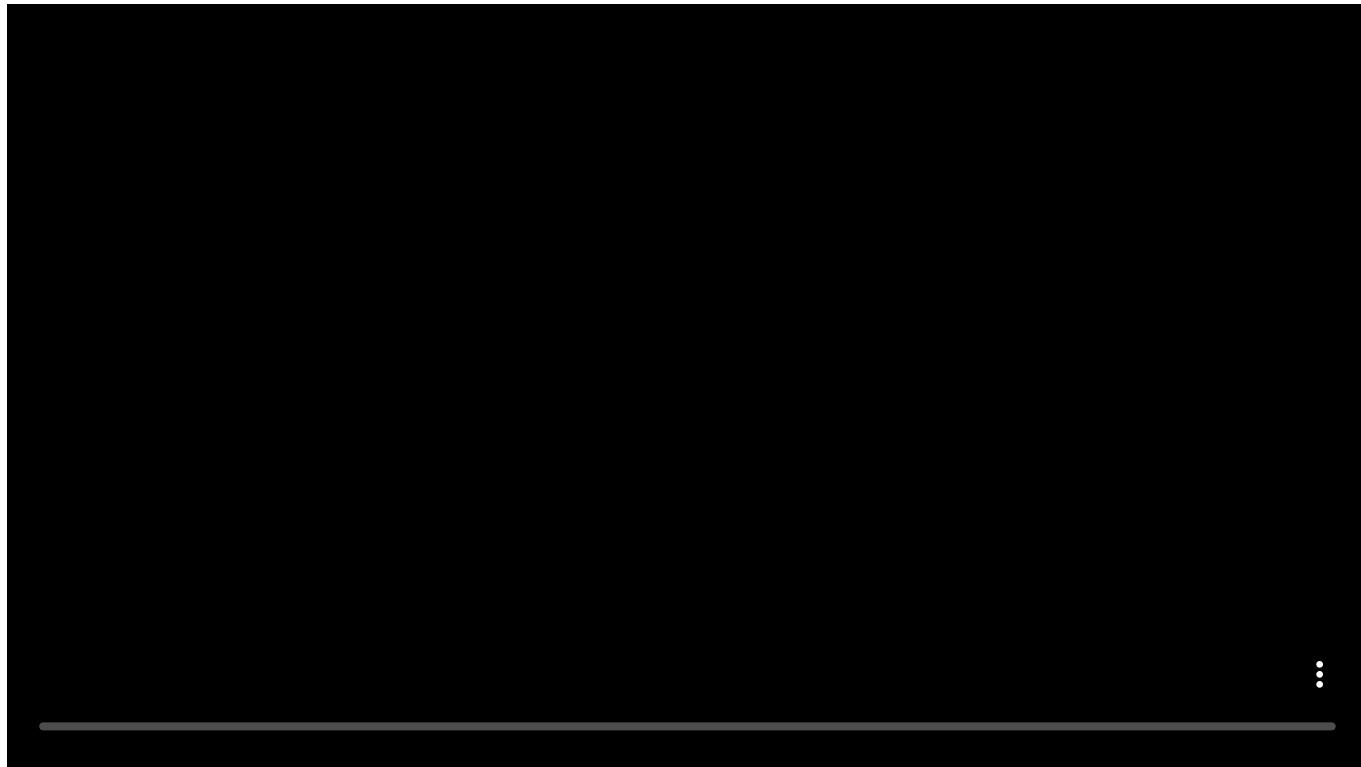
Let participants...

- choose which image they prefer
- rate images on aesthetic value
- arrange elements on a canvas
- create a pleasing or interesting stimulus
- change an existing stimulus
- ...



# Generate OCTA stimuli during the experiment

Possible with PsyNet ([www.psynet.dev](http://www.psynet.dev)) → Presentation by Peter Harrison in Methods session (May 10th, 9:00)



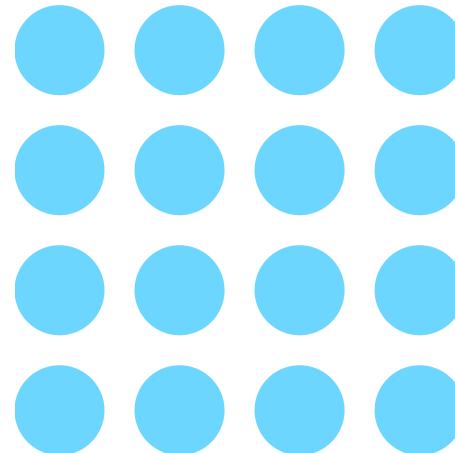
# OCTA: The future

# Future developments

- additional position patterns & deviation options
- additional animation options
- option to create multiple stimuli at once
- ...

Which additional features would you like to see?

What other feedback or suggestions do you have for OCTA?



# Cite OCTA when using it!

Always cite OCTA when using it in your work;

Van Geert, E., Bossens, C., & Wagemans, J. (2023). The Order & Complexity Toolbox for Aesthetics (OCTA): A systematic approach to study the relations between order, complexity, and aesthetic appreciation. *Behavior Research Methods*, 55, 2423–2446. <https://doi.org/10.3758/s13428-022-01900-w>

If you also want to refer to the Python toolbox or the app specifically, you can use:

- for the Python toolbox: Van Geert, E., Bossens, C., & Wagemans, J. (2021). The Order & Complexity Toolbox for Aesthetics Python library [Computer software]. [https://github.com/gestaltrevision/OCTA\\_toolbox](https://github.com/gestaltrevision/OCTA_toolbox)
- for the app: Van Geert, E., Bossens, C., & Wagemans, J. (2021). The Order & Complexity Toolbox for Aesthetics Shiny application [Online application]. [https://elinevg.shinyapps.io/OCTA\\_toolbox/](https://elinevg.shinyapps.io/OCTA_toolbox/)

# Further resources

- Read the OCTA paper
- Use the OCTA Shiny app
- Use the OCTA Python package
- Subscribe to the OCTA mailing list
- Check the OCTA documentation and example stimuli
- Revisit the OCTA workshop

my research groups, institutions, and funder



Computational Auditory Perception

MAX PLANCK INSTITUTE  
FOR EMPIRICAL AESTHETICS



Thanks to

my collaborators



Eline Van Geert

GestaltReVision, KU Leuven

Computational Auditory Perception, MPIEA

