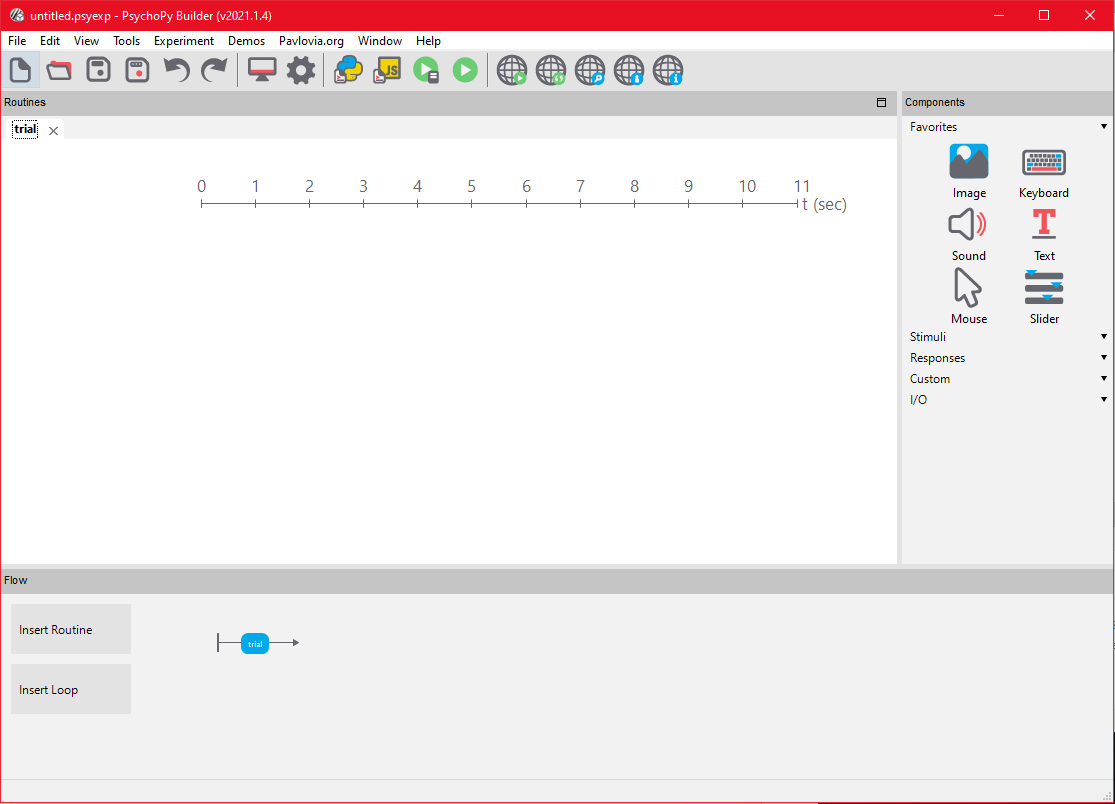
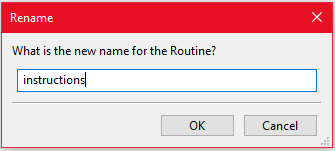
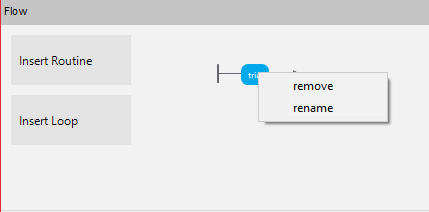
# Welcome to PsychoPy

The first thing you’ll see when you open PsychoPy is a group of windows appear. For the purposes of this tutorial, we will only be using the builder window. At the top of the builder window is the main toolbar, and the rest of the screen is separated into the **Routines**, **Components**, and **Flow** panels.



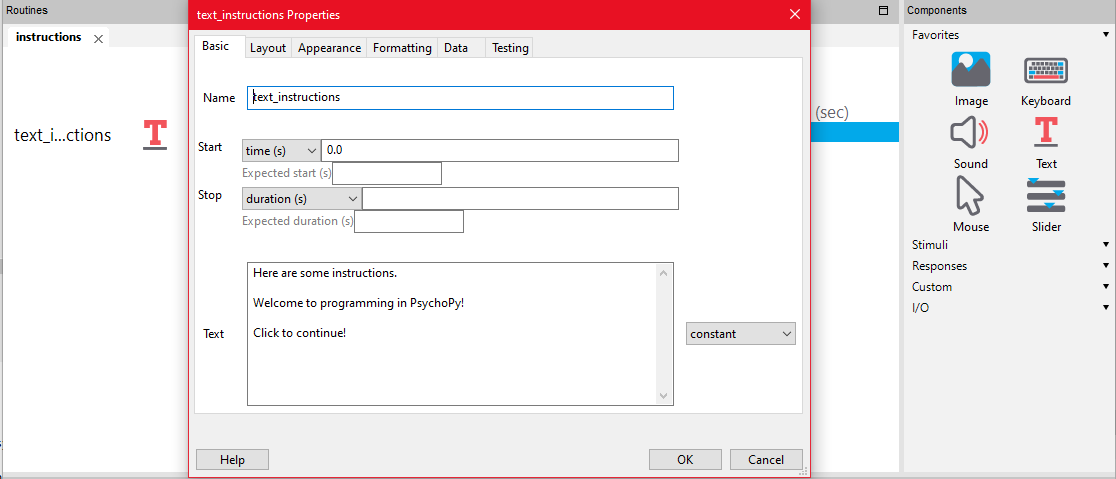
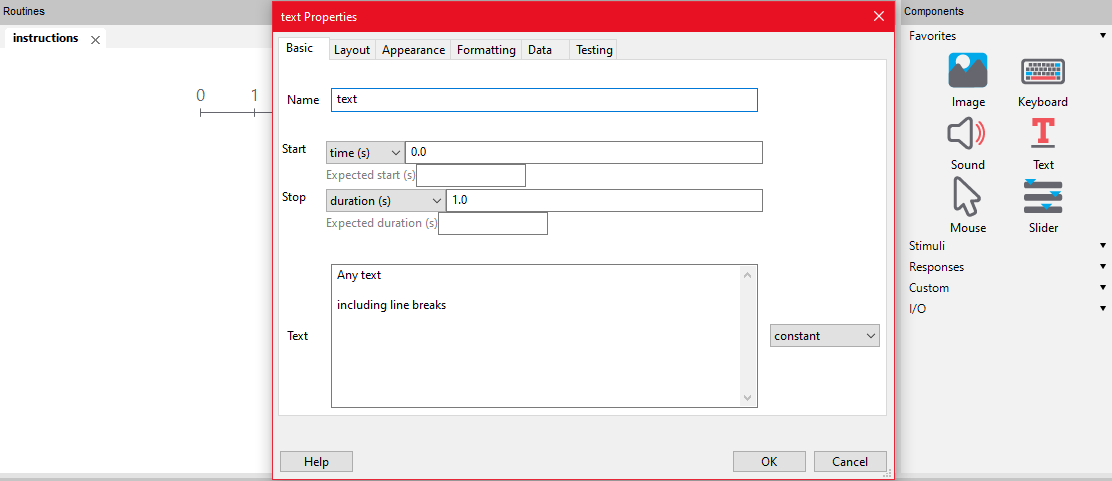
## Rename trial to instructions

At the start of our task, we want the participants to read a series of instructions. We will rename the default Routine name from “trials” to “instructions” by right-clicking on the relevant routine in the Flow panel.



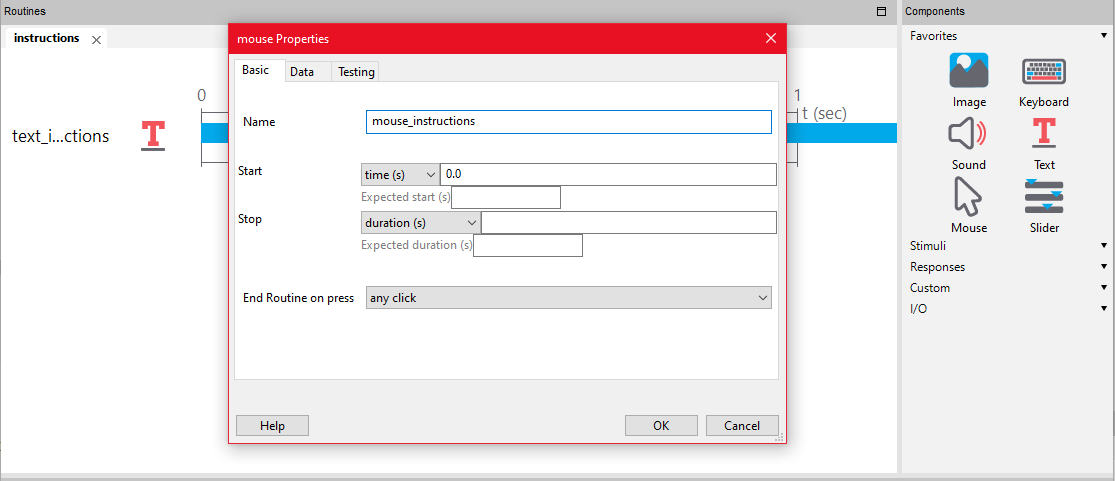
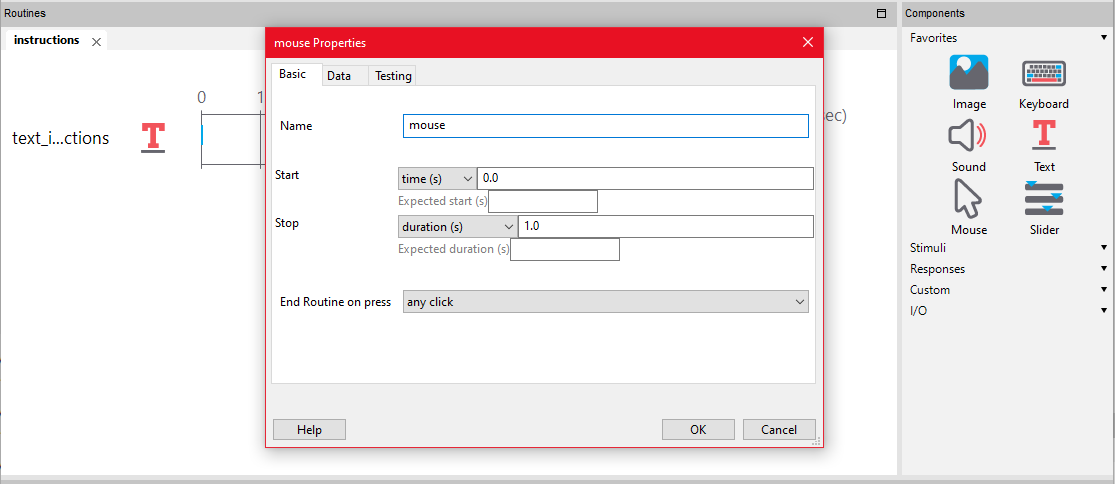
# Adding Text to Your Routine

Once you create a routine, you can add any number of components to customize the task behavior. To add a component, select the type of component you want to add in the Components panel and a blank instance of that component will be added to the active Routine. Each component has its own specific attributes and settings. In our case, we want to create a Text Stimuli Component to display simple text on the screen and greet the participant as they start the task.



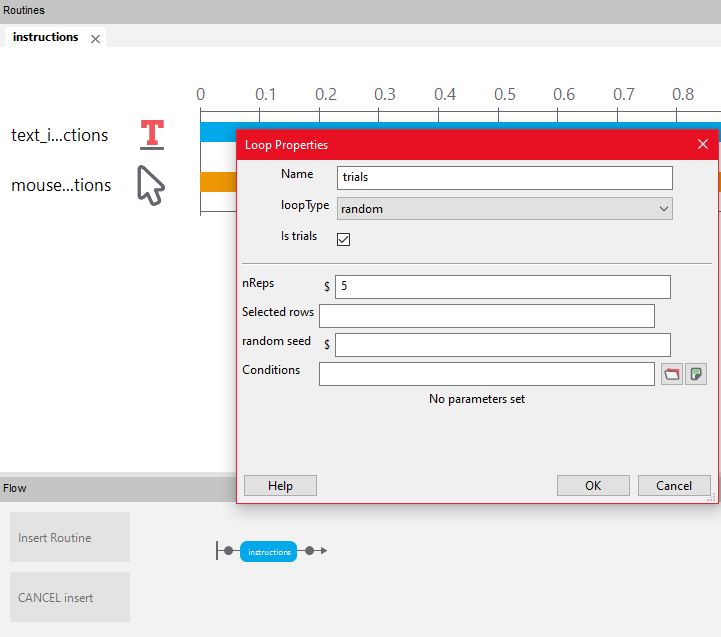
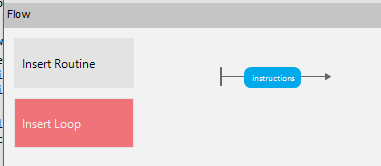
# Adding Mouse Control

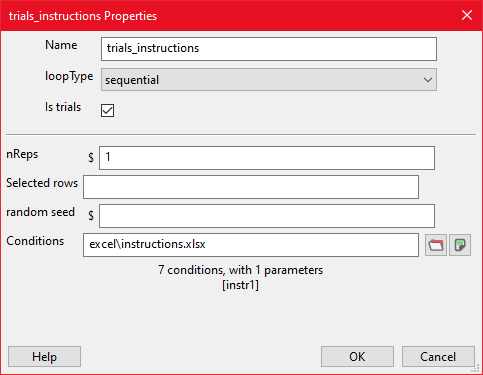
We can collect a variety of different responses from each participant and use those responses to manipulate the task behavior. In our case, we want the participant to progress through a series of instructions by clicking with their mouse. We will create a Mouse Response Component by selecting it in the Components panel and use it to End the current routine when the participant clicks anywhere on screen.

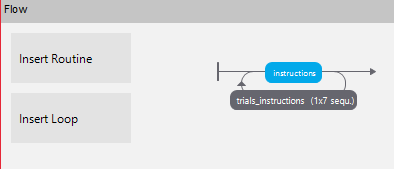


# Creating a Loop

We now have a task that provides the participant with one screen of text instructions, and a participant can press the mouse to progress through the task. If we want to add additional instruction screens, we could repeat the same process as above and create a new routine with the same components. In PsychoPy, it is better to loop Routines when possible rather than create new ones. A Loop is a specified way in which a Routine or set of Routines is repeated.



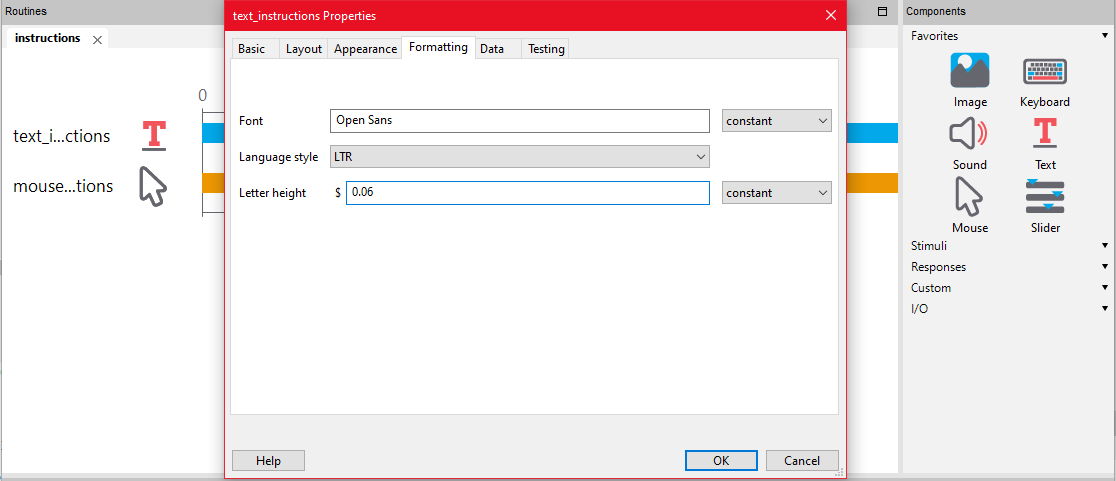
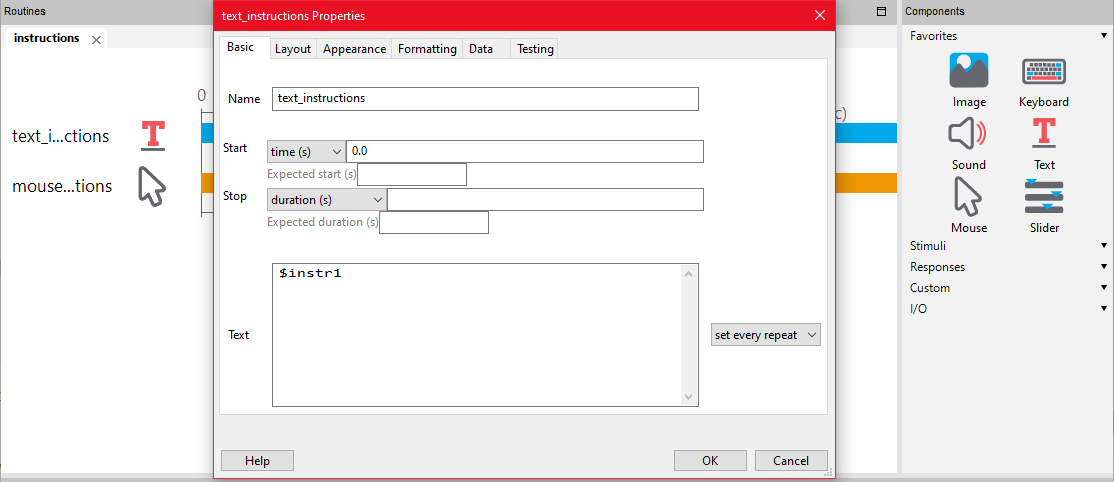




What’s happening here? We are loading a conditions table that has different parameters corresponding to different instructional text to be read (instr1). The table will be read in *sequential* order.

# Using Loop Parameters

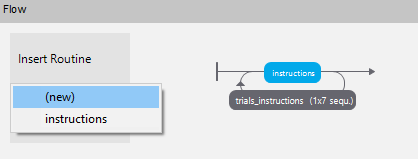
Each time the trial\_instructions loop does a pass, it will set the variable *instr1* to match the specified row in the conditions table. In order to access this parameter, we will need to modify certain components to allow them to display the variable data. In PsychoPy, variables are often denoted using a $ when outside of specific code blocks.

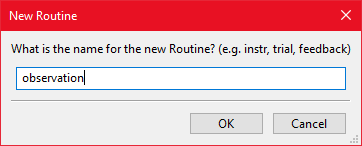


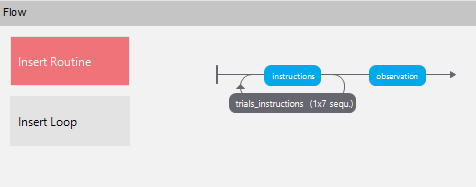
### **Please take a moment to save your work and upload the current version of the experiment as a .js file.**

# Creating a New Routine

The first part of each trial in the task is an observation phase where the participant sees the prompt and must click on it to be presented with the possible matches. We will start by creating a new Routine, and inserting it after the instructions phase.

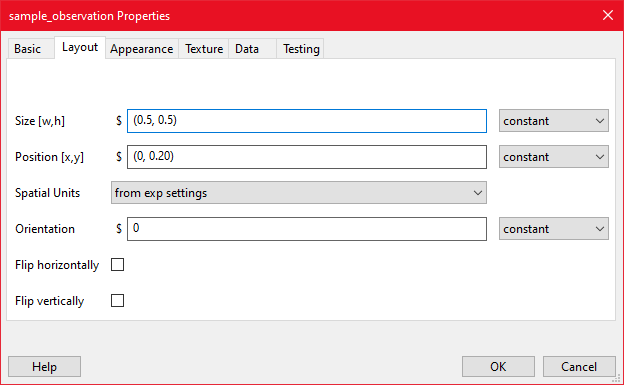
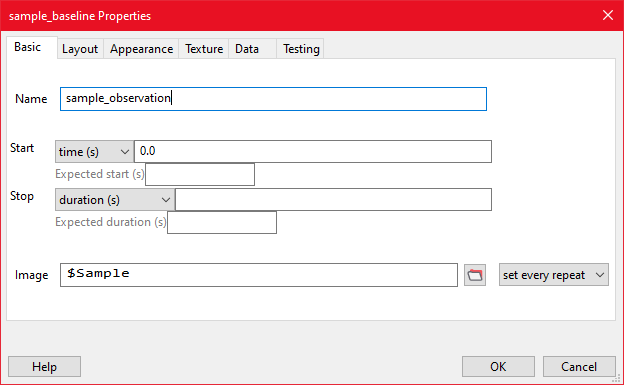
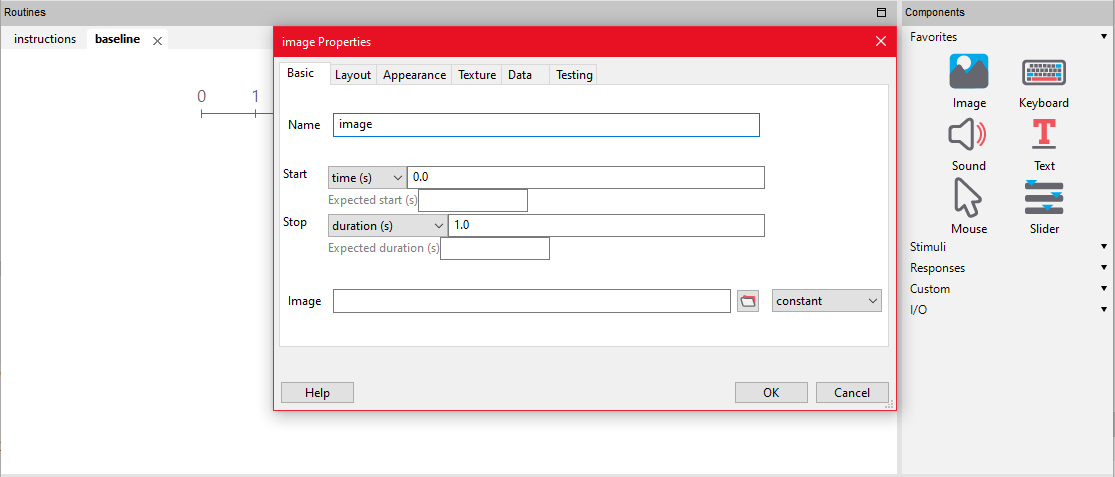






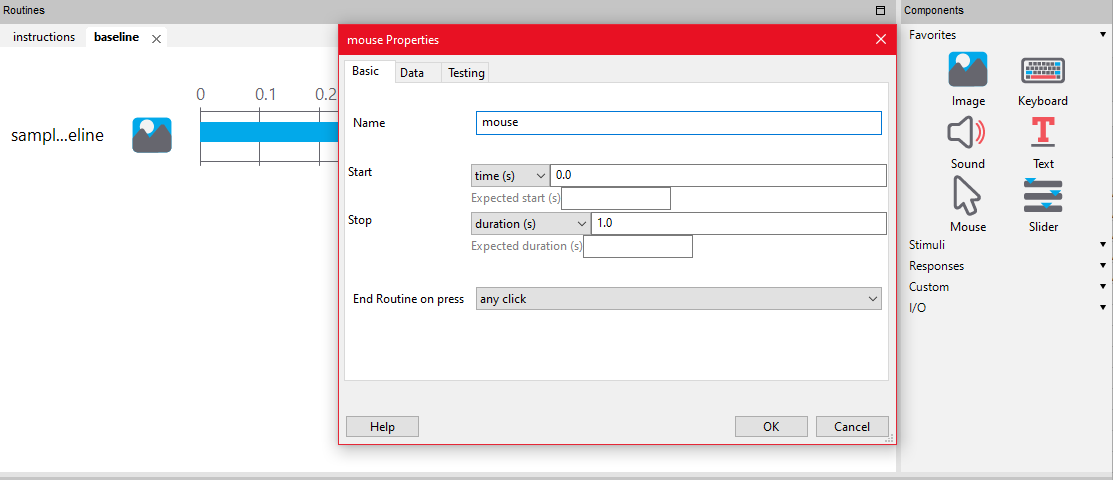
# Adding Image Stimuli

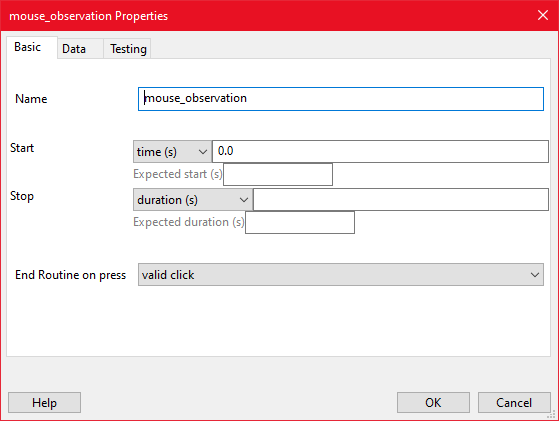
We will create an Image Stimuli Component that will display the sample stimuli. By utilizing loops we can manipulate what stimuli are presented during each trial. The image is set to a variable that will match the parameters of a loop we will create in the next few sections. The field next to the image filename describes when and if the image value will be updated. Since we will refresh the image every trial, we will have it set every repeat.

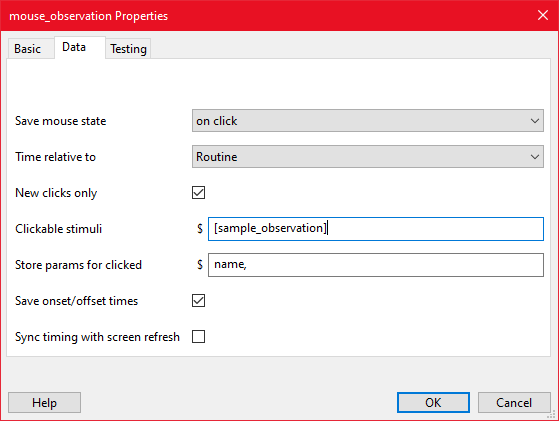


# Clickable Stimuli

When a participant makes a click in a routine that contains a mouse component, the resulting data is saved according to the settings of the mouse component. By manipulating a few settings, we can create a trial that only progresses once the participant clicks a valid target. We will modify the mouse component settings of *End Routine on Press*: to “valid click” and add the sample target to the currently-empty list of clickable stimuli.

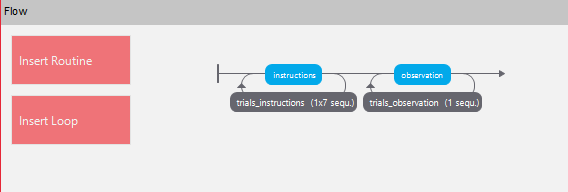
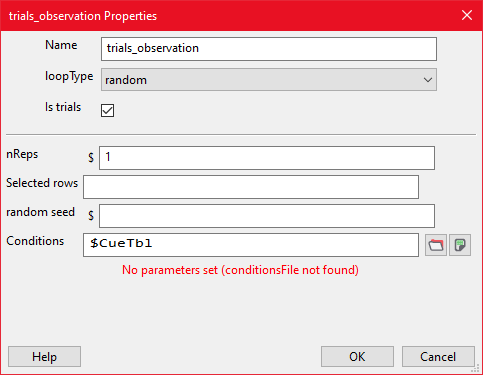
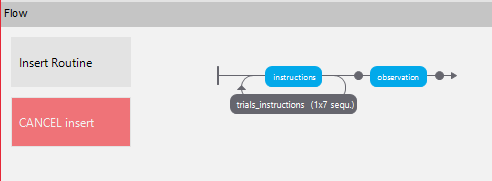






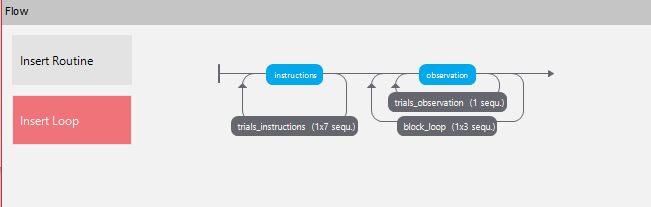
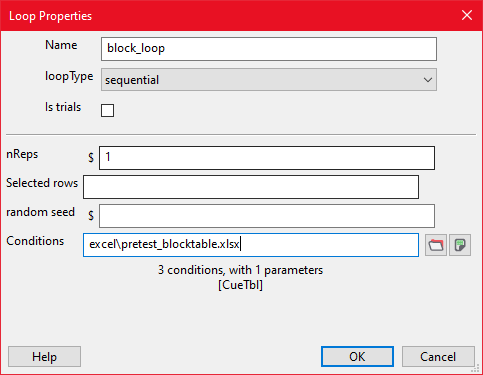
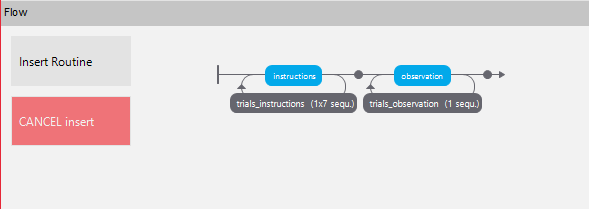
# Looping Task Conditions/Creating Block Loops/Nesting Loops

We will create a loop to control the observation routine trials. It is common for researchers to want different segments or blocks of a task to have modified characteristics, allowing for maximum usage of existing routines while simplifying organization and minimizing redundancy. First, we will create an inner loop that will control the trial-by-trial behavior.



## Block Looping with Block Table

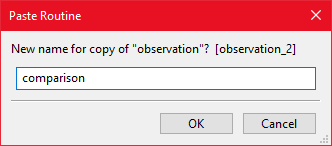
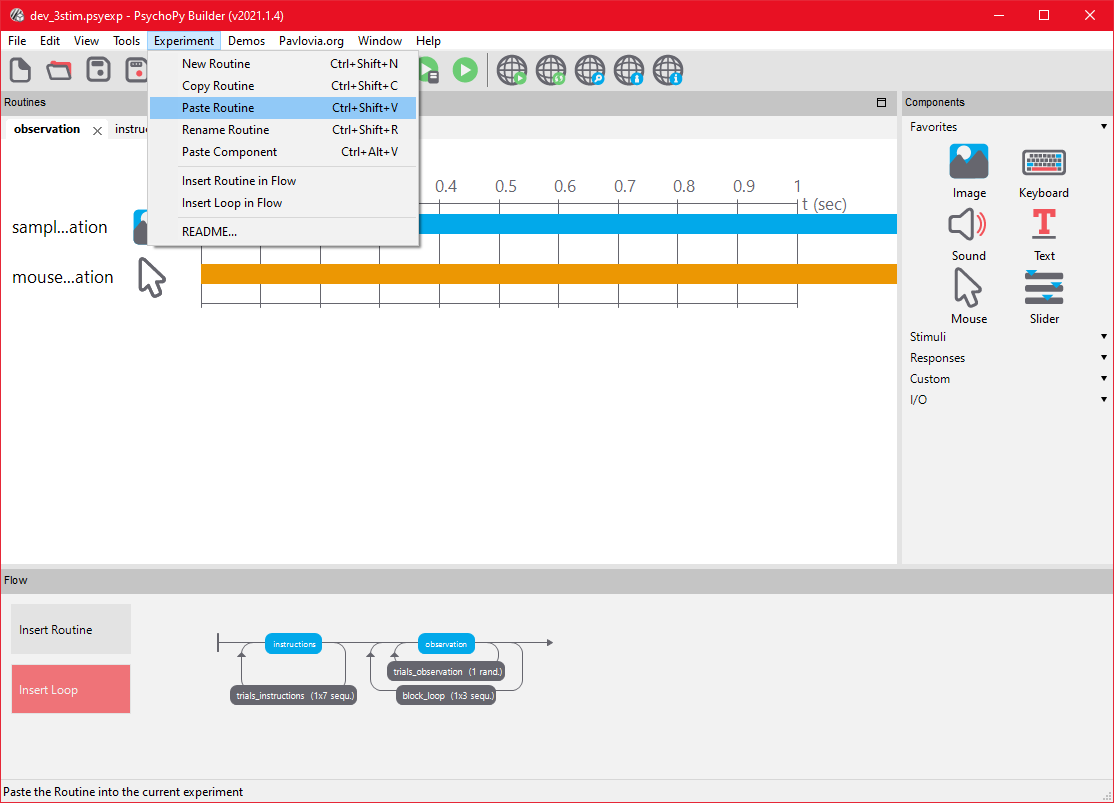
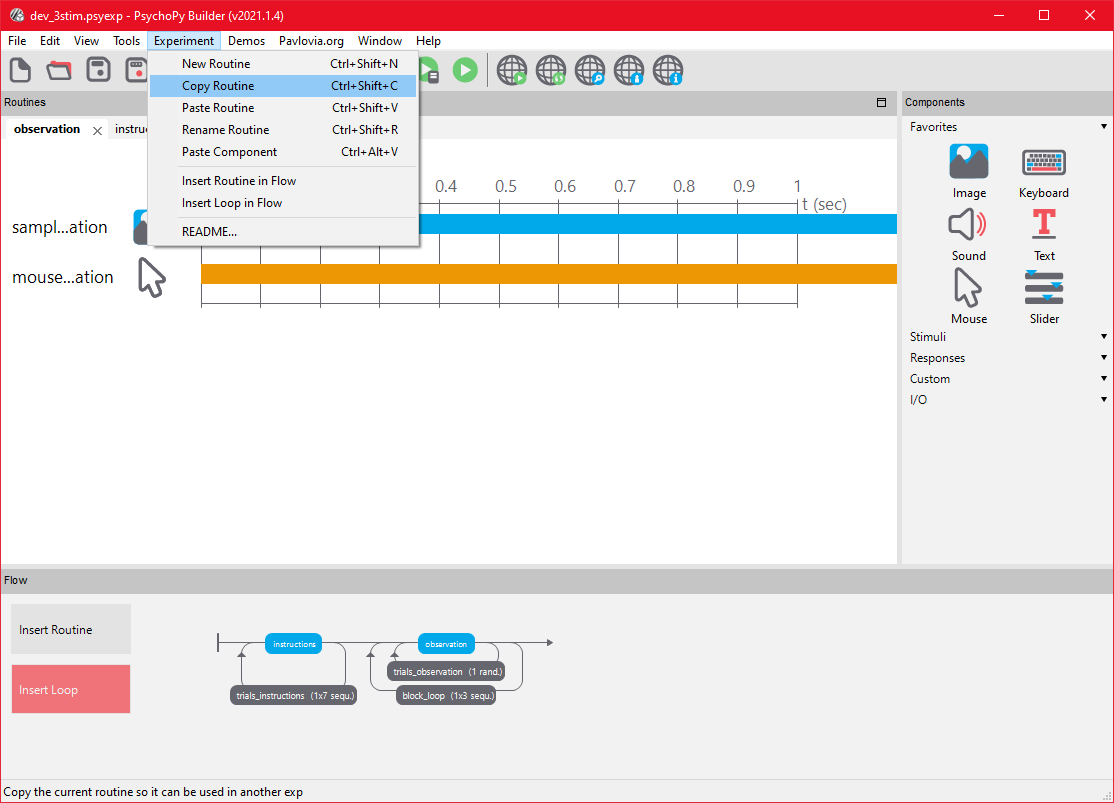
In the last loop, we created an inner loop that would draw its parameters from a defined *CueTbl*. By turning the conditions table into a variable, we will be able to modify what conditions table will be used during each block. We will use a nested loop structure, or a loop within a loop. This will provide is with easier control of our program by utilizing defined conditions tables. The outer loop will have different properties compared to the inner loop, but the logic behind both is very similar.

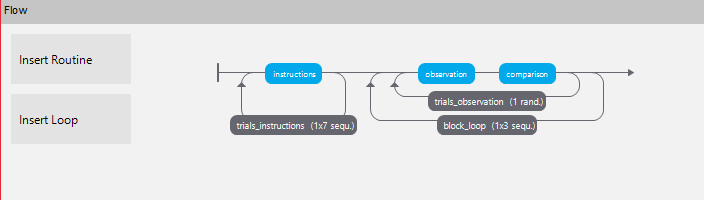
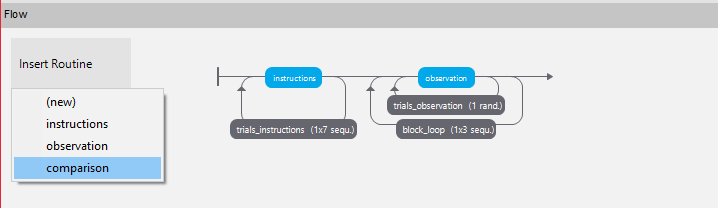


### **Please take a moment to save your work and upload the current version of the experiment as a .js file.**

# Copying and Modifying a Routine

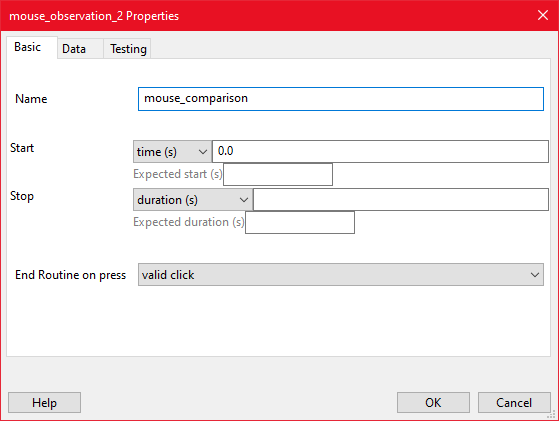
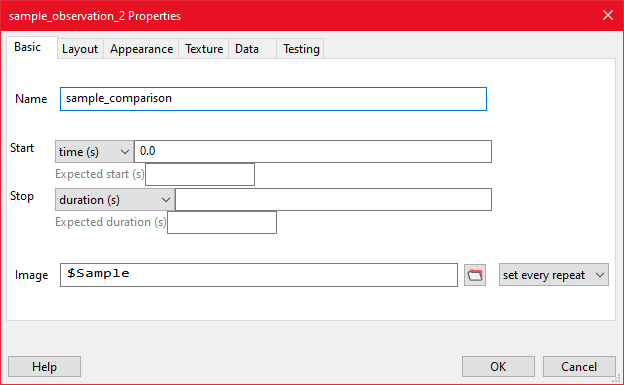
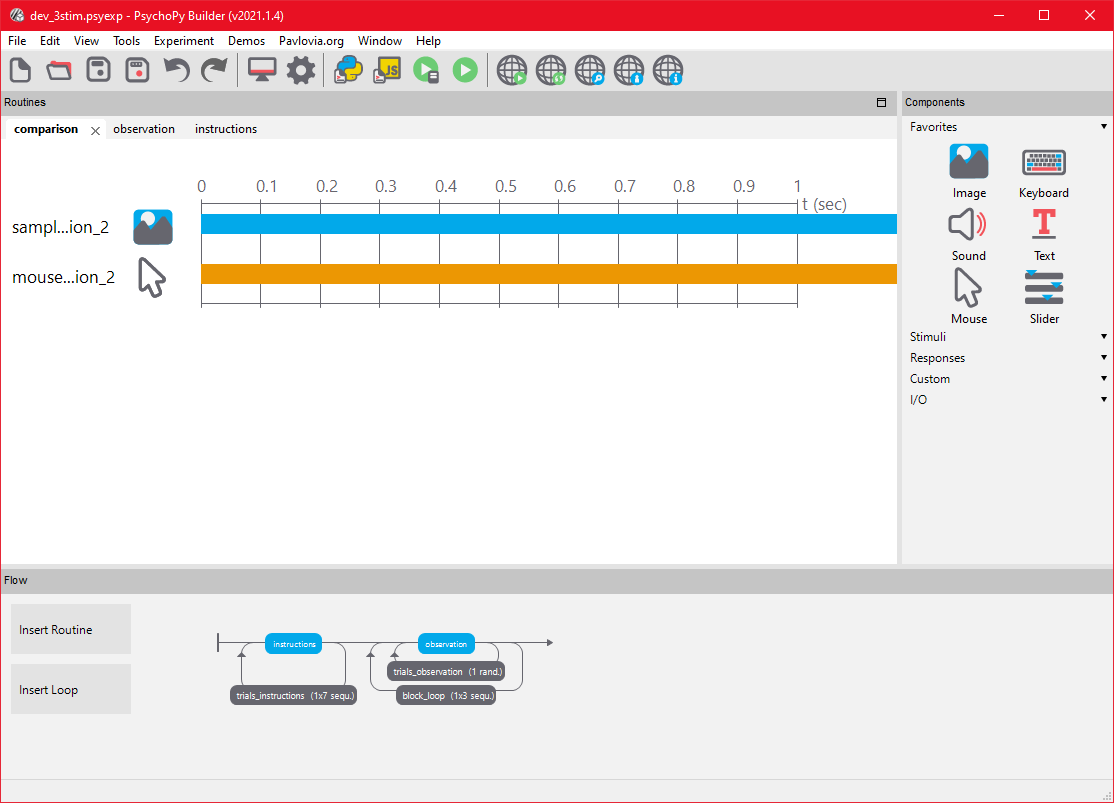
Following the observation phase, we want to present participants with 3 potential matches to the sample stimuli. Rather than creating a new routine, we can save some time by copying and pasting the routine we had previously worked on. Each of the components we added before will be transferred over to the new Routine.





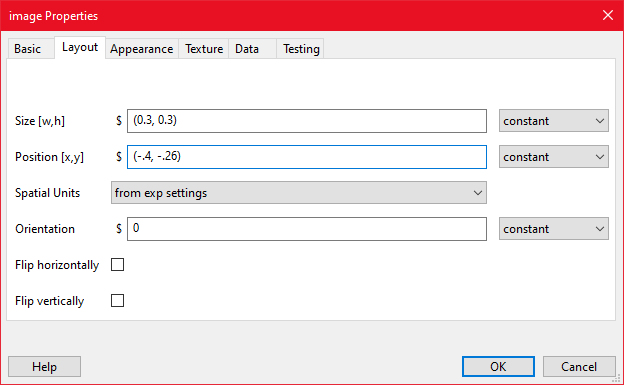
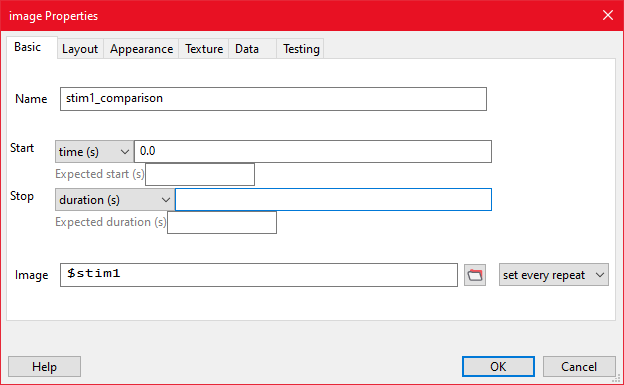
## Naming Modifications

Now that the routine is copied, we should change the name of the variables to differentiate between the phases of the trials. This will help simplify data collection and analysis.

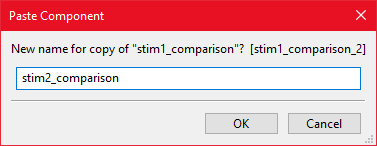
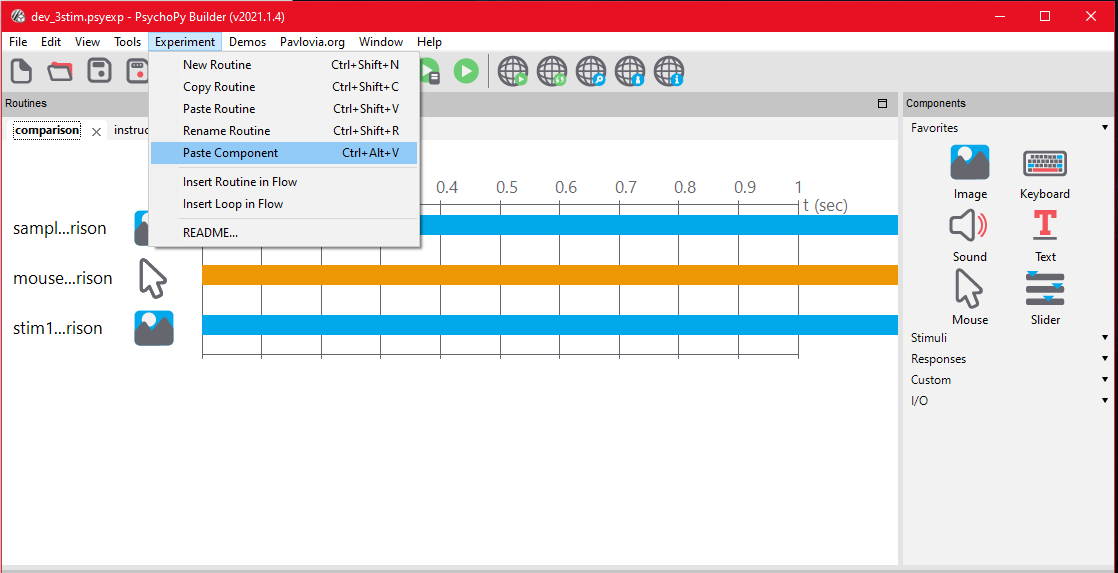
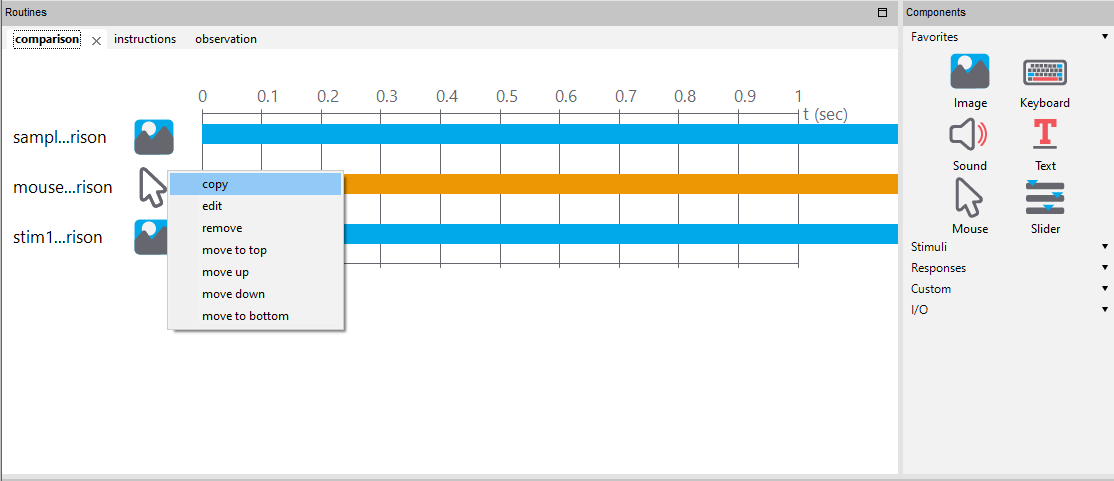


* New Match Stimuli

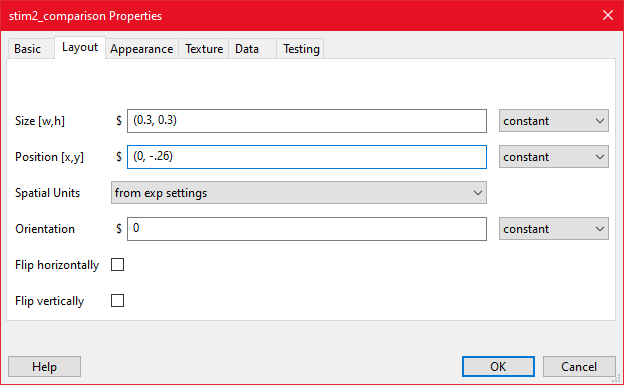
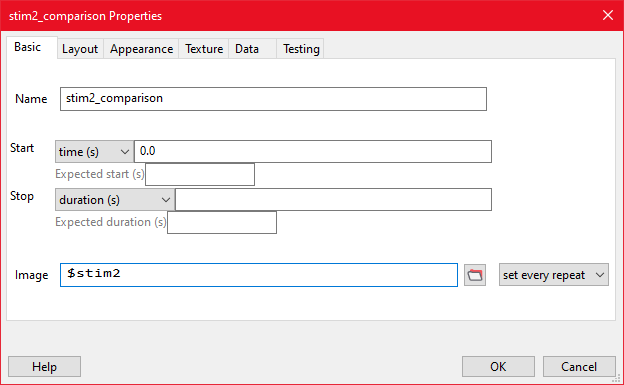
Now we will define the potential matches to the specific sample. Each of the stimuli presented will be controlled using loop parameters drawn from a conditions table (CueTbl). We will create the first match stimuli with the settings listed below:



# Copying Components

Similar to Routines, we can also copy and paste components to save some time. We will utilize this to create two more match stimuli with similar properties to that of the image stimuli we just created. We will need to modify the name and position of the component once it has been copied.

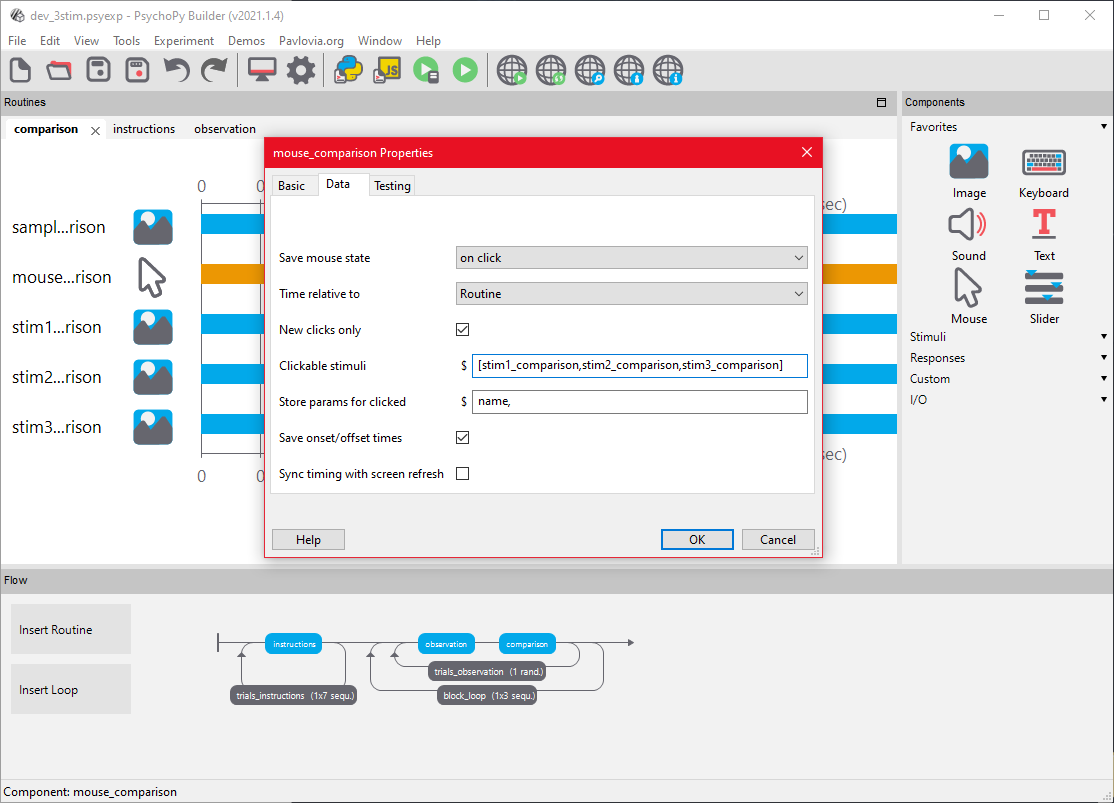
-modifying once copied:



\*\*\* For Stim3, follow same steps above but replacing the following: (1)Name:stim3\_comparison (2)Image:$stim3(3)Position[x,y]:(0.4,-0.26)

## Set Valid Targets as Clickable Stimuli

Lastly, we will define these stimuli as clickable targets and remove the sample from being a valid target during this Routine by modifying the properties of the existing mouse component.

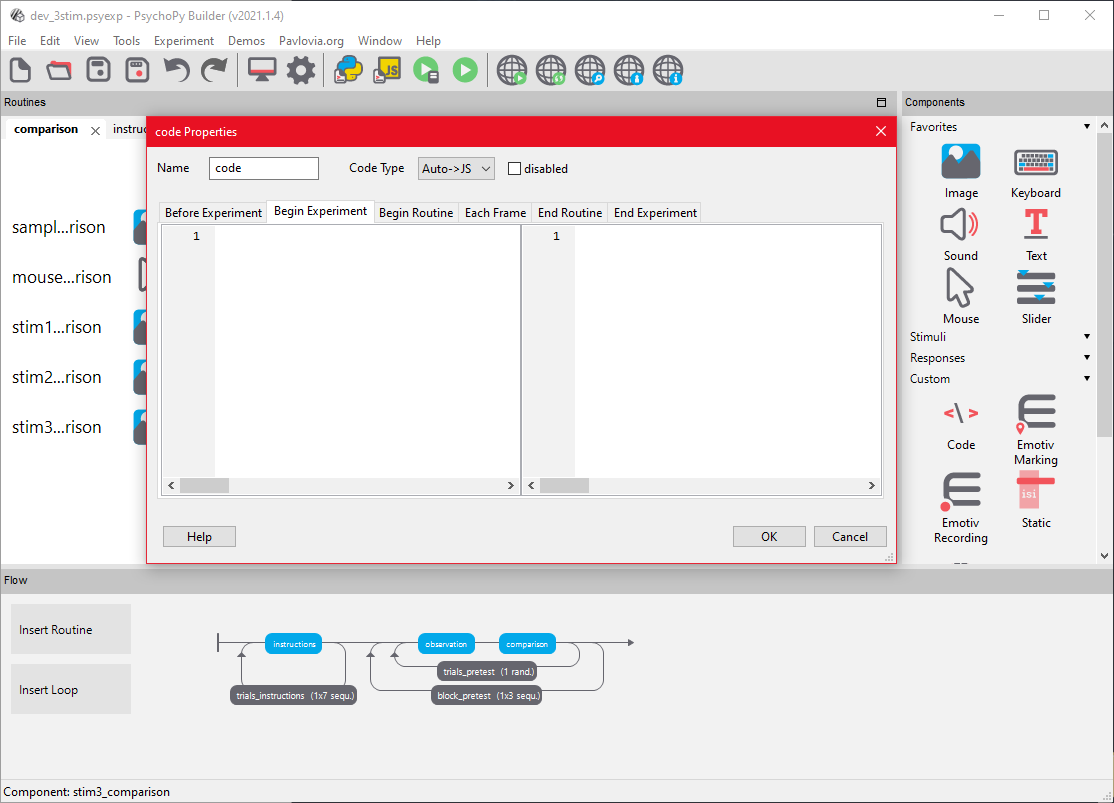


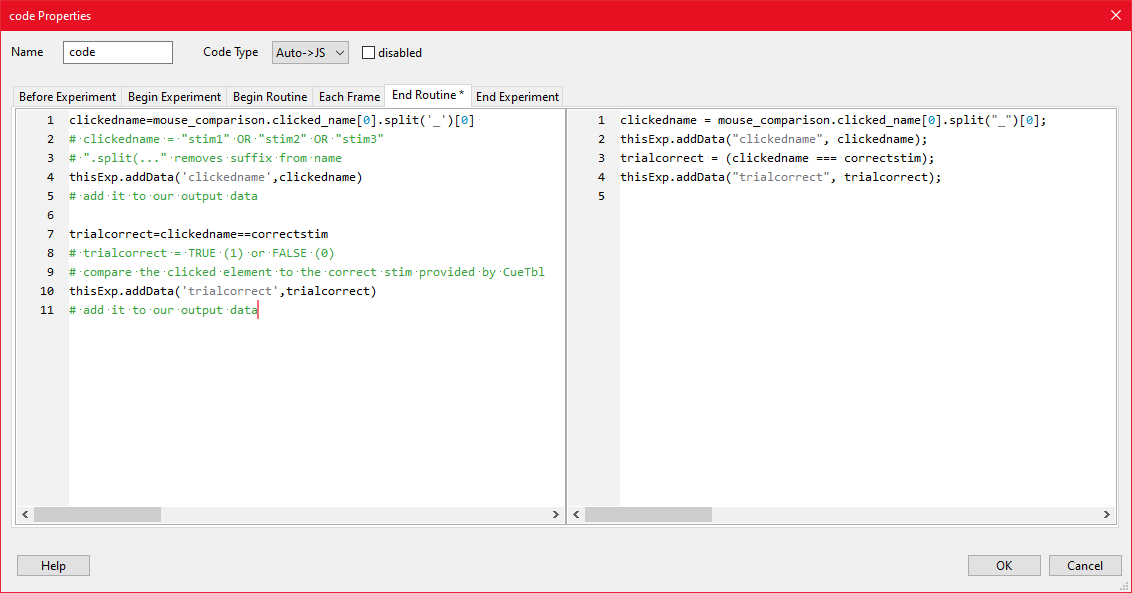
# Recording Data and Manipulating Tasks using Code Blocks

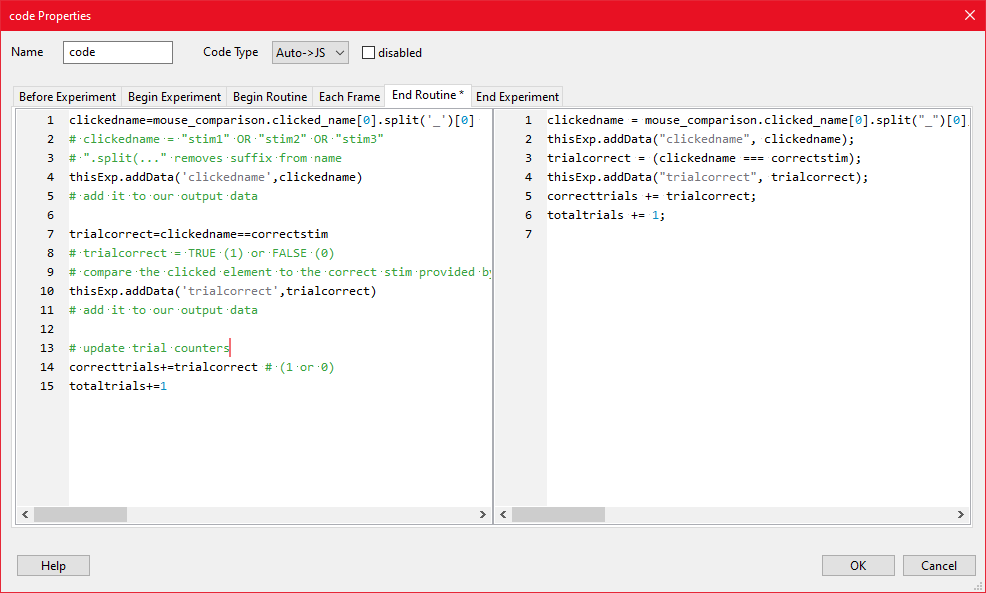
There are a lot of ways to manipulate stimuli and data using only a combination of the tools we’ve already used. For more advanced features, most researchers utilize code blocks in order to create specific task behavior or customize how and what data is collected. The primary language used in PsychoPy is Python, which is used for client-side development. Once the task is fully developed, it can be automatically translated into JavaScript and then uploaded onto the web. The steps required to upload an experiment to the web, and debug JavaScript issues, are outside of the scope of this tutorial.

**Refer to supplementary code documents for information on what code should be placed in which routines and the timing of each code snippet.**

New Code Component in Comparison Routine



The tabs at the top of the Properties window denote when the code will be run during the experiment.

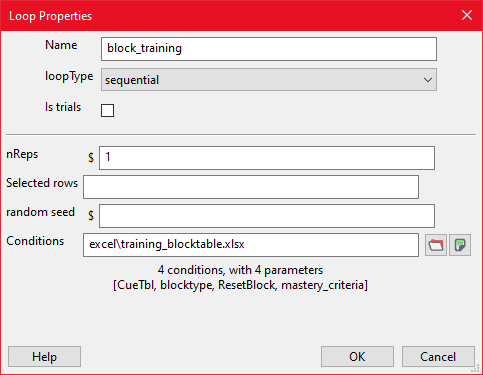
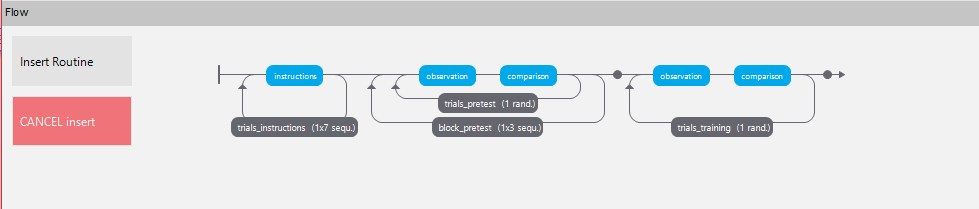
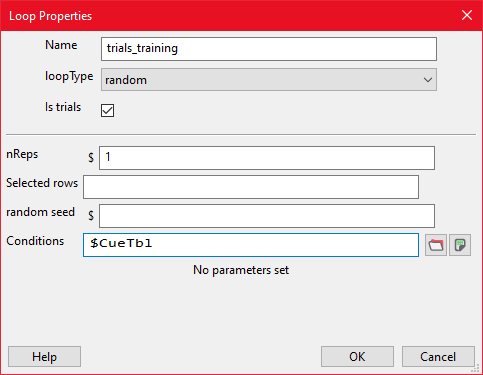
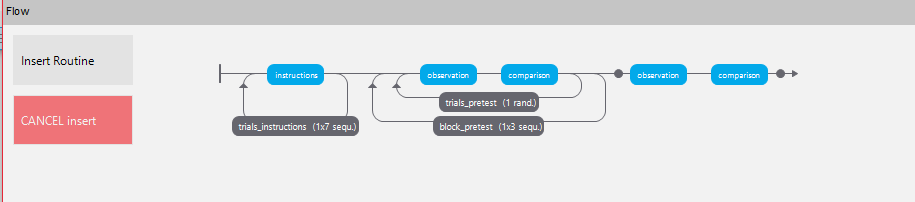
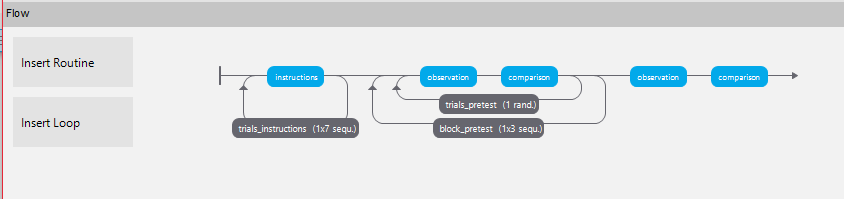
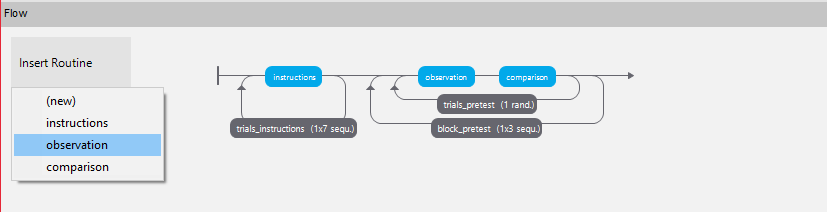


What’s this code doing? Each time we go through this routine, we want to pull out the name of the stimulus the participant clicked on and save it. Then, we can check to see if the name of that clicked stimuli matches the name of the correct stimuli as provided in the CueTbl., and then add that data to the output. Finally, we can utilize counters to keep a running tally of how many correct and total trials the participant has completed.

### **Please take a moment to save your work and upload the current version of the experiment as a .js file.**

# Create Training Blocks

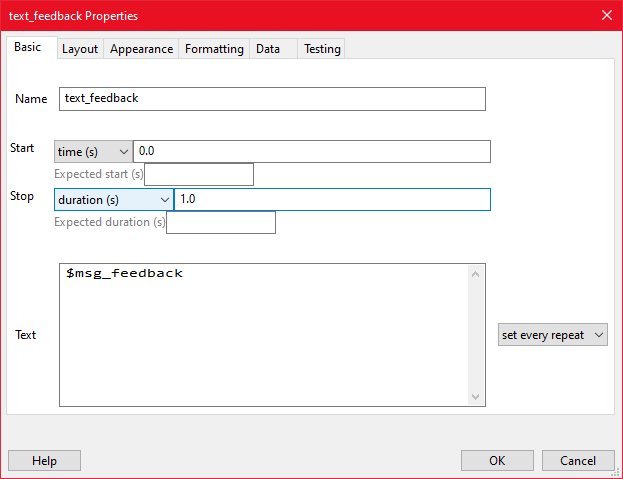
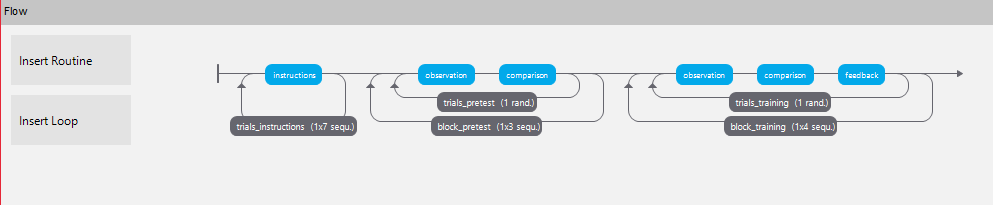
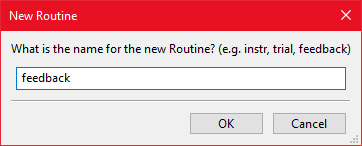
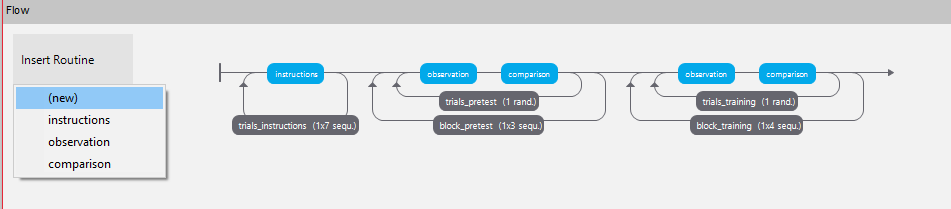
In our task, we want to have specific testing and training blocks. The only difference between these two blocks will be the presentation of feedback. On training blocks, participants will receive text feedback so they can correct themselves, whereas testing blocks will not feature feedback. With the testing phase already constructed, we will utilize a similar structure to start creating the training blocks. We will copy the same routines utilized earlier, and create loops with very similar parameters.



excel/training\_blocktable.xlsx

# Adding Trial Feedback

As described before, we want the training trials to feature feedback after each trial. We will create a new routine that features a text component with a variable message. This message will be set in the code block of the preceding Routine, *comparison*. We will also utilize string formatting to display variable data (number of correct/total trials) as part of text on the screen.

Graphical user interface, text, application, email

Description automatically generated

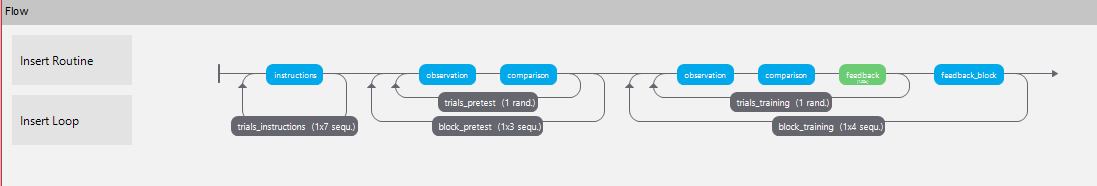
### **Please take a moment to save your work and upload the current version of the experiment as a .js file.**

# Mastery Criteria and Block Feedback

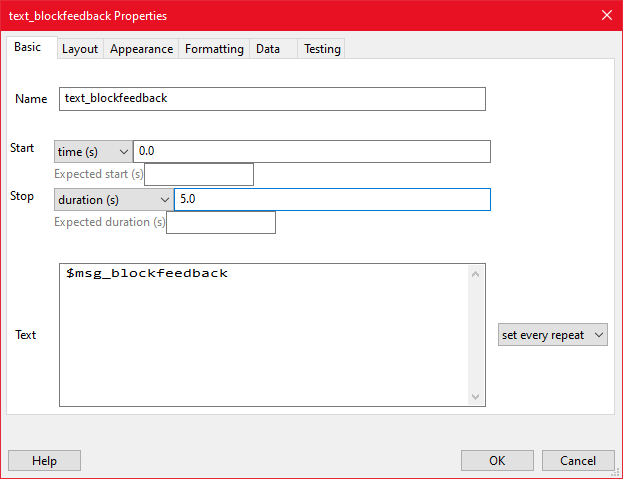
As part of the task, participants will pass or fail each block based on whether the reached a specified “mastery criteria” that is defined in the condition tables. This will determine how accurate participants need to be in order to pass a block. For now we just want to tell participants if they reached criteria or not during their latest block. Later, we will utilize this setup in order to cause specific changes to task behavior and ordering of the Routines.

Graphical user interface, application

Description automatically generated



Add text component to block\_feedback routine.



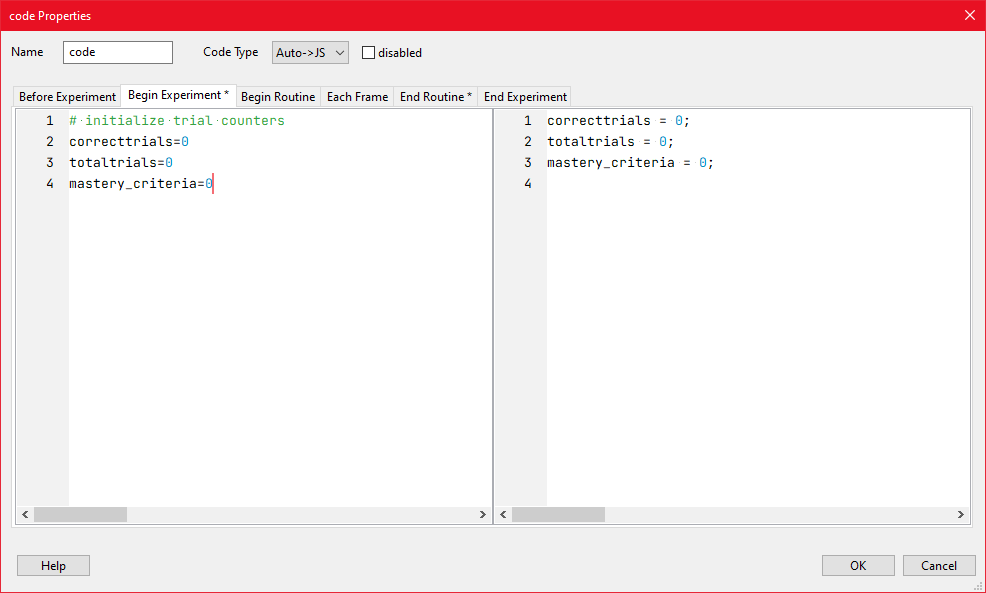
Go back into **COMPARISON** routine and add code component

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

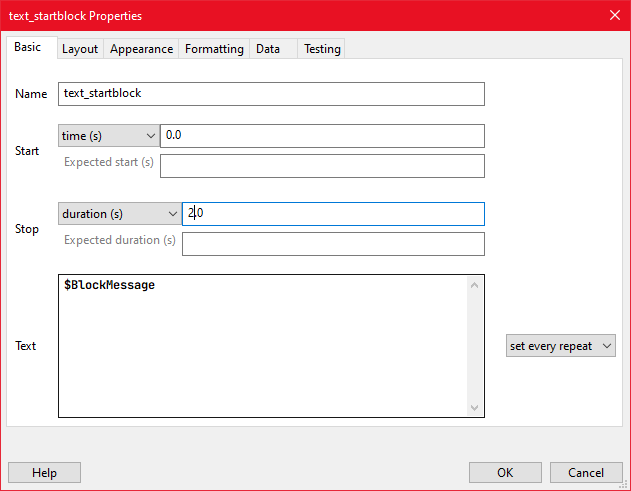
Description automatically generated

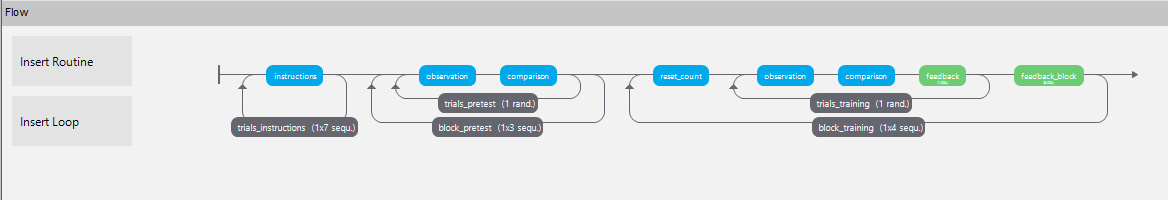
* Also need to change the **Begin Experiment** code

## Reset Counters when entering new loop

We also want to give participants a heads-up that they’re entering a new block. We will create a simple routine that displays information about what type of block the participant is entering, and whether or not they will receive feedback.

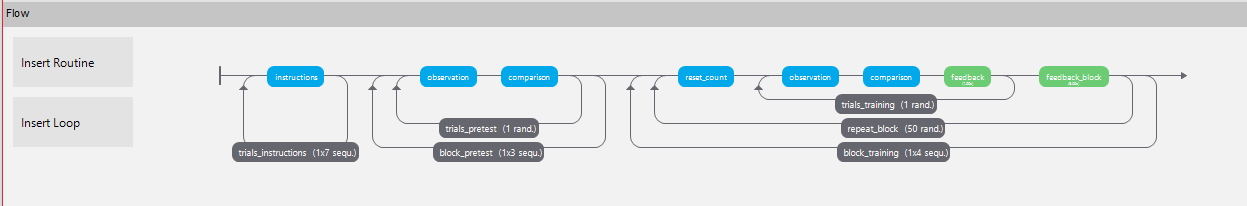
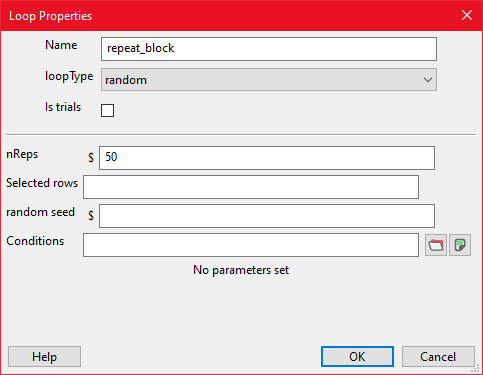
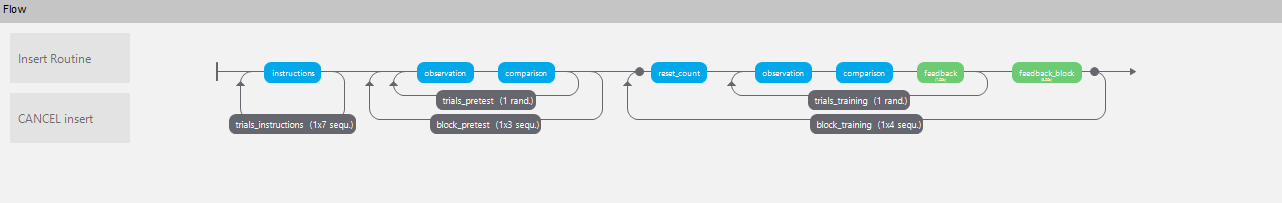
* Graphical user interface, application

  Description automatically generated
* Graphical user interface, text

  Description automatically generated
* 

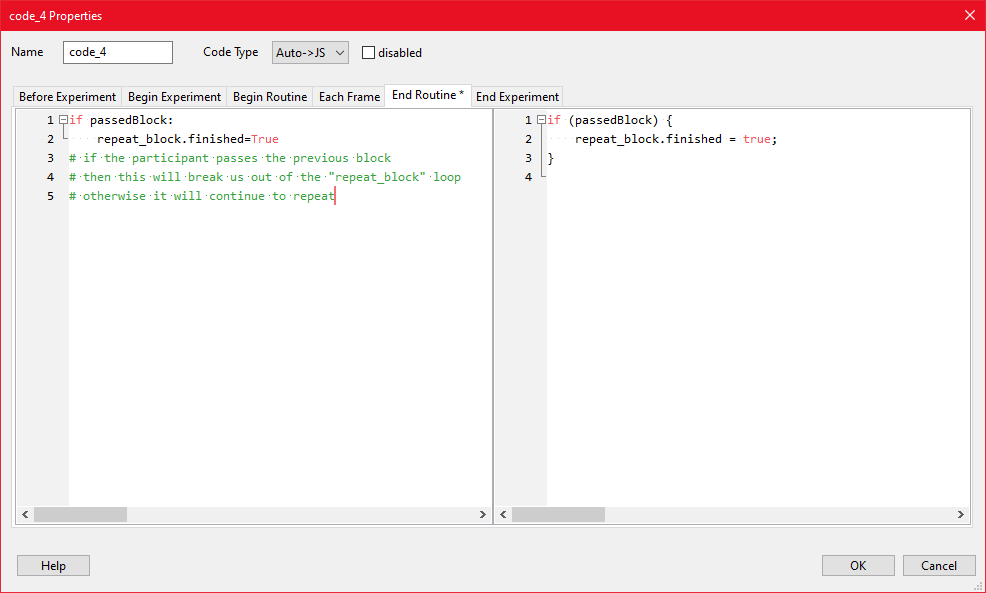
# Repeating Block after Failing

In our task, we want the participant to repeat a training phase if they do not meet the mastery criteria for that specific block. To do so, first we will encapsulate the relevant task in a containing loop that repeats indefinitely, or to some arbitrarily high about (50). Once that is created we can define different ways to exit the loop.



Now we must create a way for the task to progress if the block is passed, otherwise the participant will be stuck in the loop forever. To do this, we add an explicit exit within the code component in the block\_feedback routine. When the code is run and the routine ends, the repeat\_block loop will break and the participant will continue on with the task.

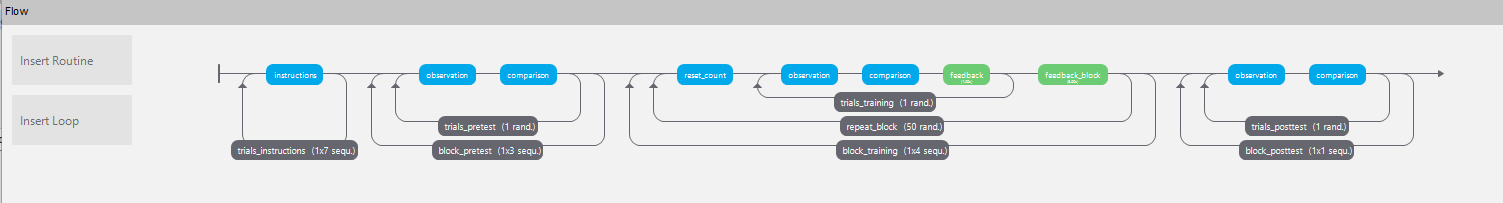
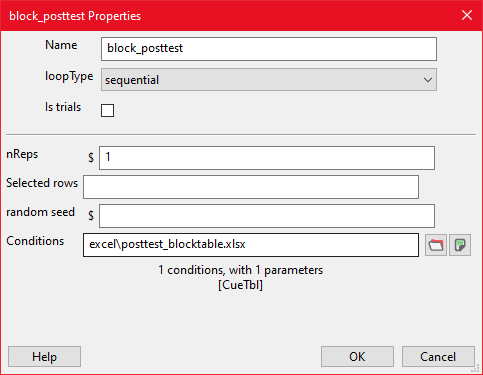
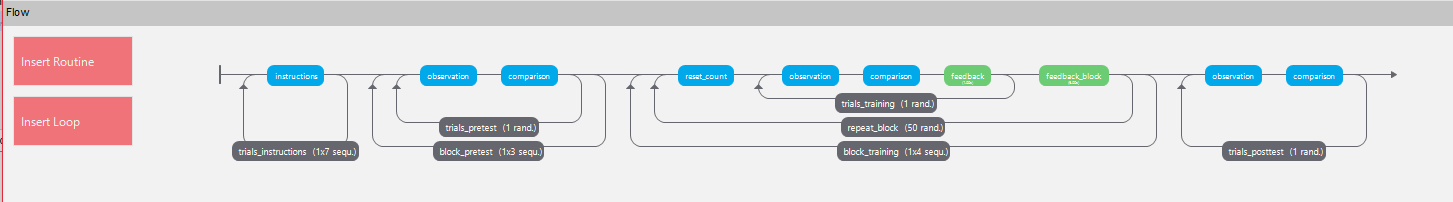
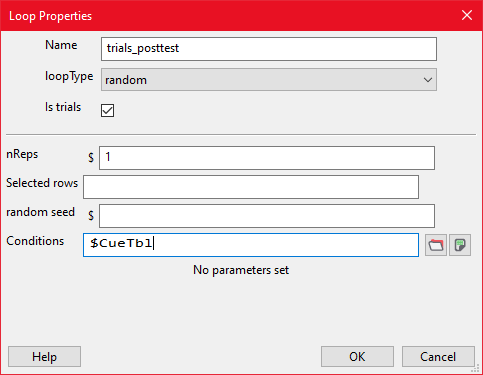
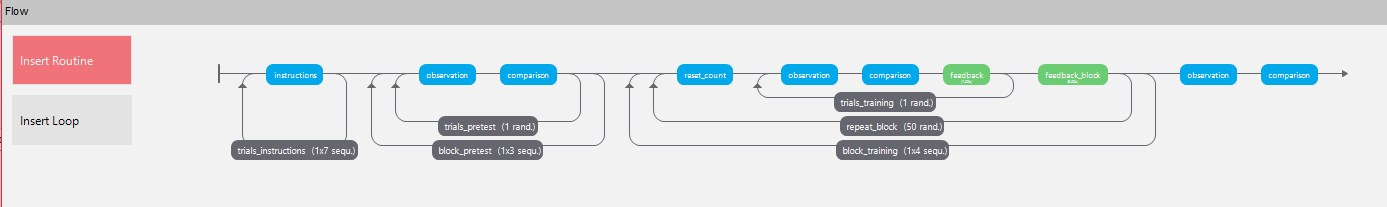
Add new code component to block\_feedback routine.



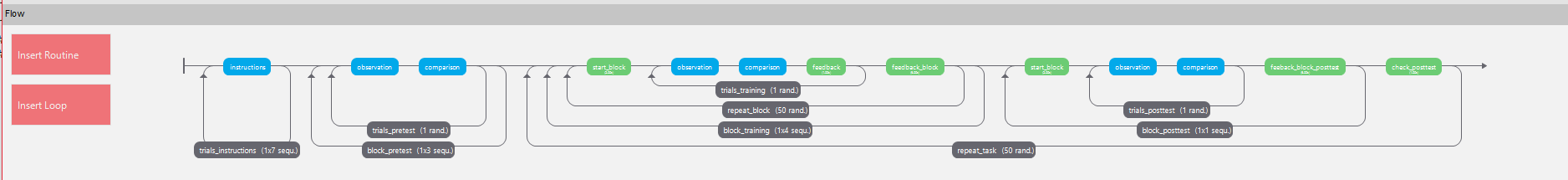
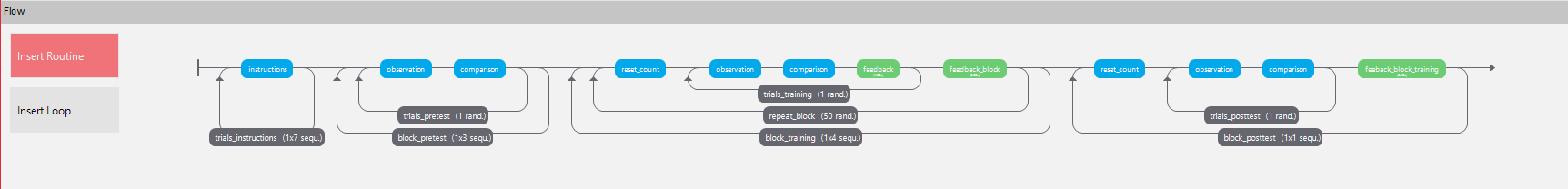
### **Please take a moment to save your work and upload the current version of the experiment as a .js file.**

# Advanced Looping/Returning to Specific Routine on Failure

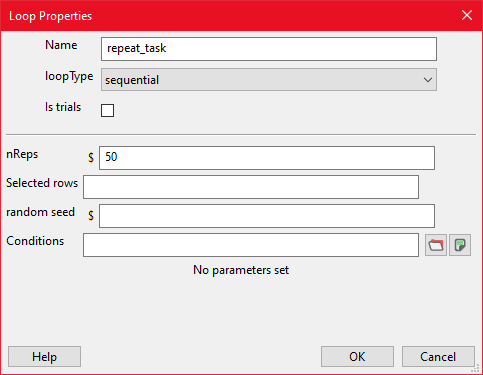
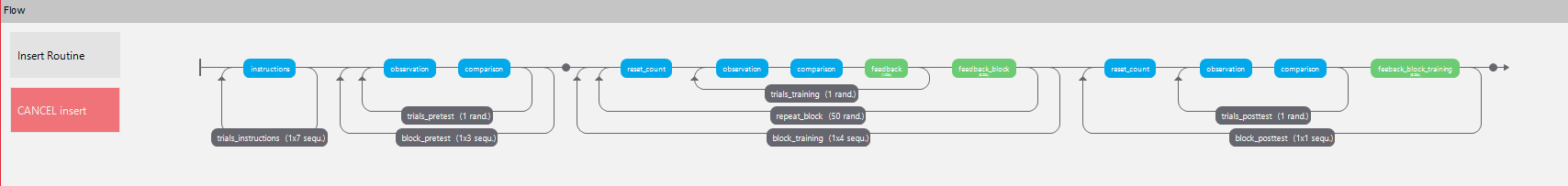
Lastly, we want to set up the post-test phase of our task. Our desired behavior is that if the participant fails any of the post tests, they will be sent to the start of training and be required to go through each of the training blocks before arriving back at the post-test phase. Similar to before, we will start by inserting existing routines and creating loops to control trial and block behavior.

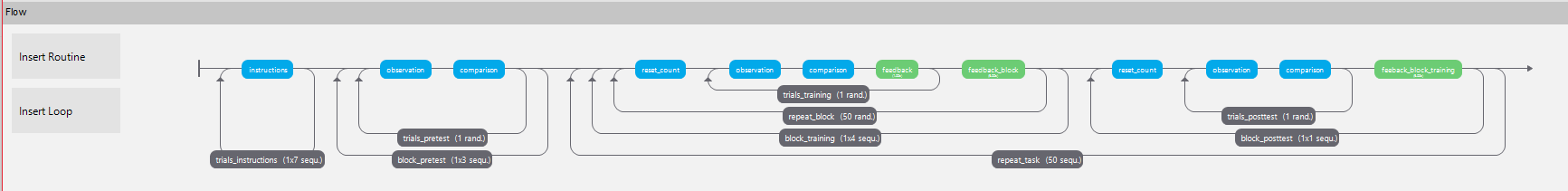


* Copy block\_feedback Routine and modify it
* Copy the routine using the steps outlined in **Copying and Modifying a Routine**, name it “block\_feedback\_posttest”

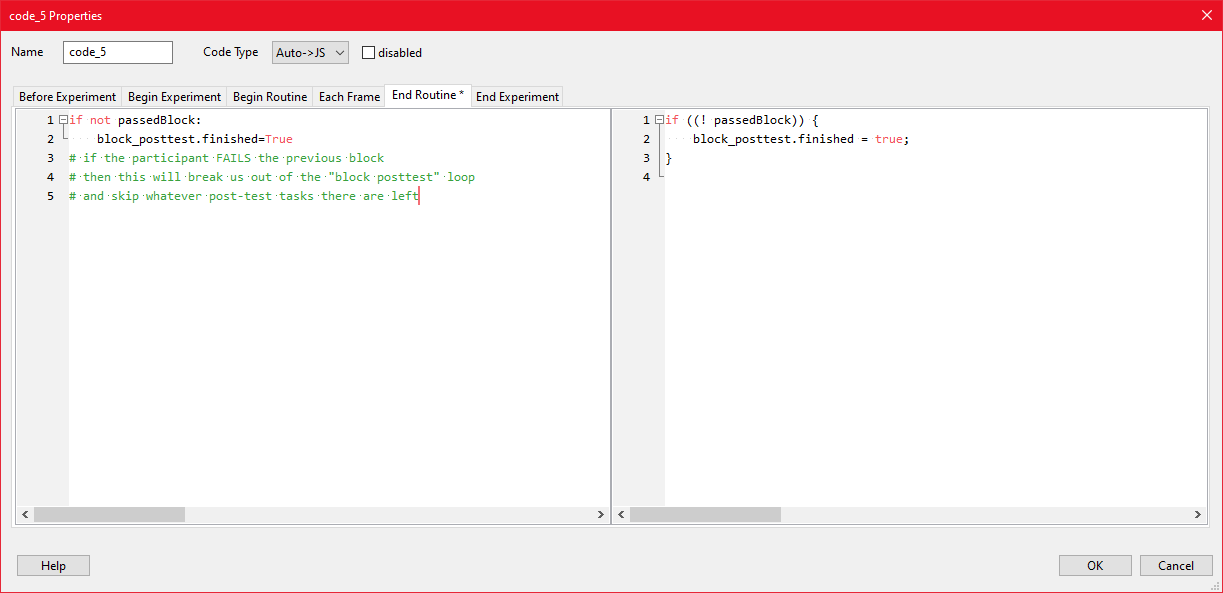


* Add control loop to make the task repeat when the participant fails after a post-test block.

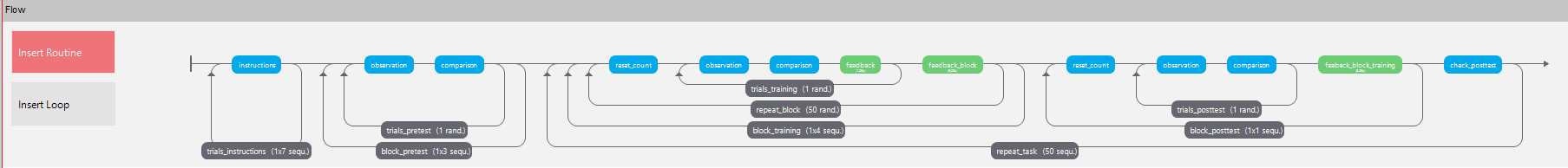




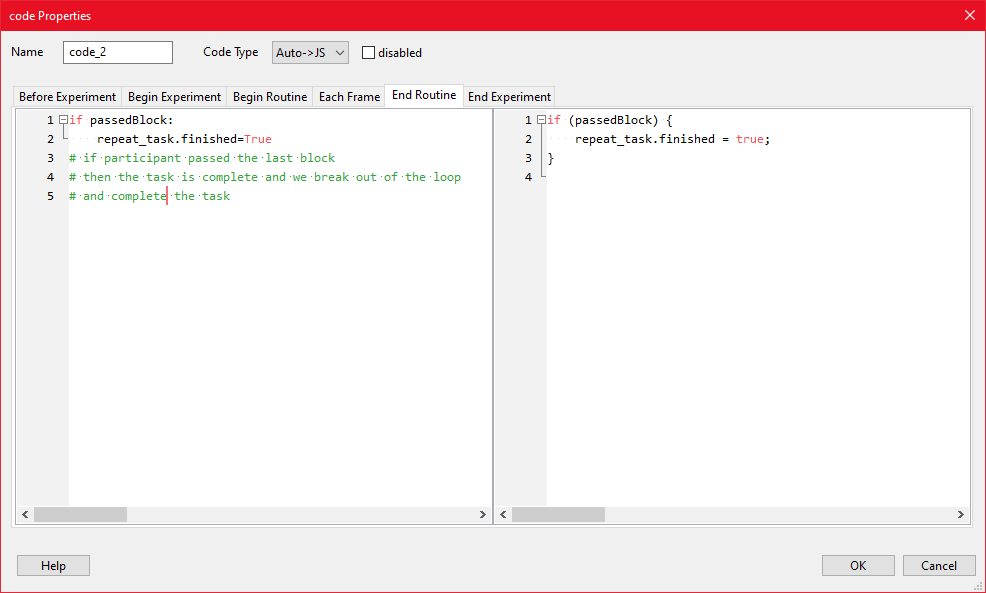
* Modify code component in the block\_feedback Routine

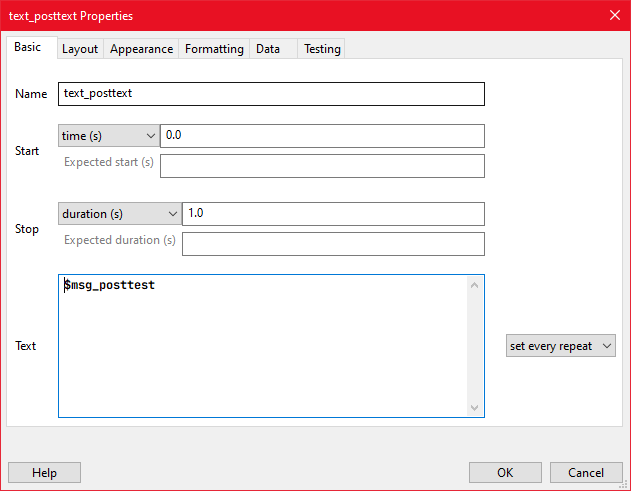


* Add new routine to allow us to end the repeating loop



* Add code component to that routine with this code:





* Modify code component of block\_feedback\_posttest

