jsPsychR manual

Gorka Navarrete

2022-07-31

Contents

Chapter 1

What is jsPsychR?

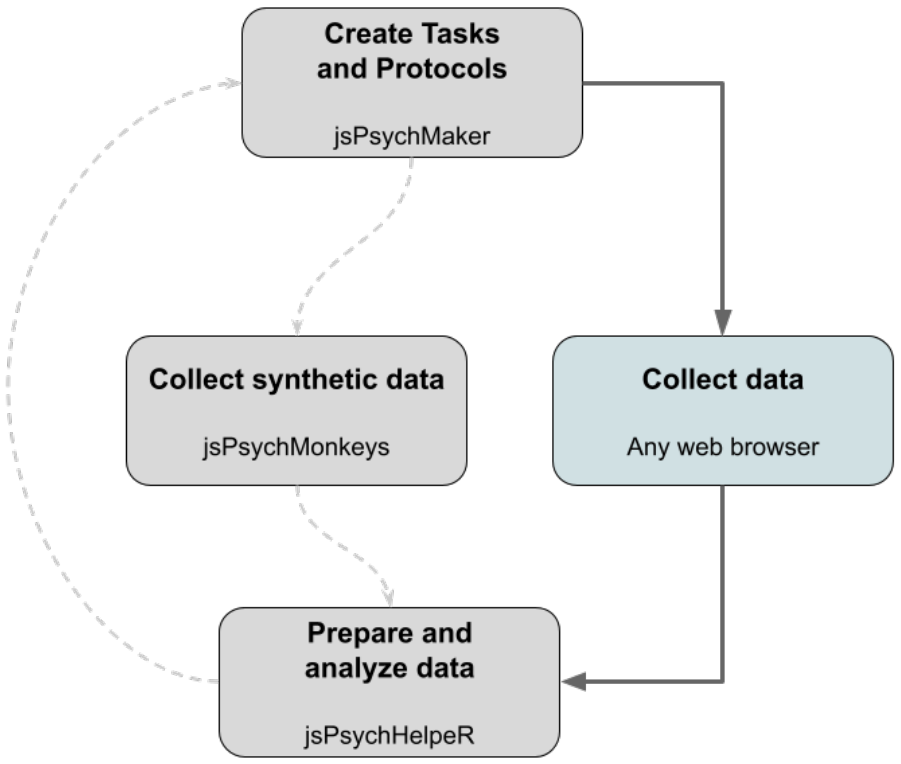
`jsPsychR` is a group of tools to help create experimental paradigms with jsPsych, simulate participants and standardize the data preparation and analysis.

We have three main tools:

- jsPsychMaker: Create experiments with jsPsych, randomize participants balance between conditions, etc.

- jsPsychMonkeys: Release monkeys to a jsPsych experiment using the R package {targets}, docker and {RSelenium}.

- jsPsychHelpeR: Standardize and automatize data preparation and analysis of jsPsych experiments created with jsPsychMaker.



Contributors

- Gorka Navarrete
- Herman Valencia

Chapter 2

Reproducible experiments

We use different technologies to develop experiments. Some examples are Psychopy, Qualtrics, Limesurvey, jsPsych, Gorilla, etc. Each of these has advantages and disadvantages and, in general, there are pragmatic aspects to take into account when adopting one or the other (cost, type of experiment - EEG/behavioral, lab/online -, lab history and available resources, . . .).

We opted mainly for jsPsych to run behavioral experiments because it is an open source javascript library, based on standard web technologies, and can be used online and offline.

In the last years, we started working on a set of tools to create jsPsych experiments (jsPsychMaker), simulate participants (jspsychMonkeys) and standardize and automatize the data preparation and analyis (jsPsychHelpeR).

Our final goal is to have a big catalog of tasks available to use in the jsPsychMaker repo. Each of the tasks should run with jspsychMonkeys to create virtual participants. And each task will have a sister script in jsPsychHelpeR to fully automate data preparation.

2.1 Open and reproducible experimental pipeline

To replicate an experiment from a publication is not trivial. One of the main goals of this sytem is to be able to create, share and reproduce an experiment, its data, and data preparation and analysis without any extra effort.

Furthermore, all the components of the pipeline should be Open Source, which allows reviewers, collaborators, etc. to check and run the code. This also makes it accessible to anyone with a computer and access to the internet, eliminating cost constrains.

With this system you can create a paradigm, simulate data and prepare data and analysis almost automatically (including anonimization).

The system output is standardized, so names of variables and the structure of the data are predictable. Finally, the plots, tables, reports and analysis are reproducible, so you can get enerything ready with simulated data, preregister or even better, go for a registered report and just relaunch the data preparation and analysis when the participant's data arrive, with a single command.

2.2 Automatization

We tried to make a few basic things right, but this is an evolving project, and some things are more complex than one would want. Please do report the issues you find:

- jsPsychMaker issues
- jsPsychMonkeys issues
- jsPsychHelpeR issues

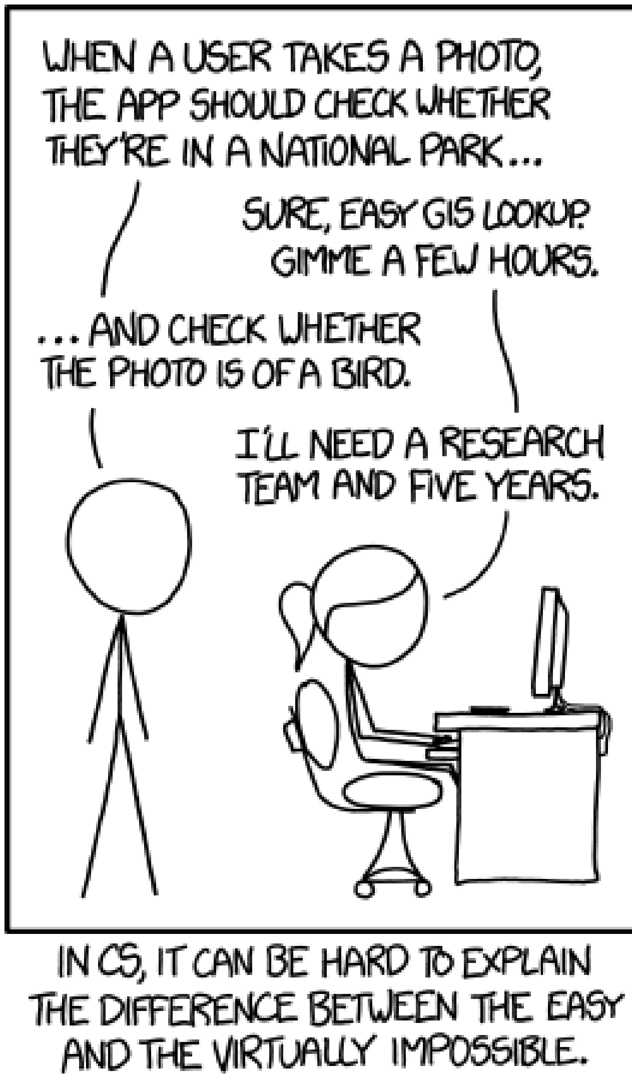

WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Figure 2.1: SOURCE: https://xkcd.com/1425/

Chapter 3

Quick Guide

3.1 jsPsychMaker: Create an experimental protocol

See the jsPsychMaker chapter for more detailed instructions.

Outline

1) Download jsPsychMaker
2) Go to the folder `canonical_protocols` and edit the `config.js` file to select the tasks you need
3) Open `index.html` in your browser

---

1) Download jsPsychMaker

Open RStudio and run the following two lines in the console. This will download the last stable jsPsychMaker version `v0.2.0`. When you finish, a new RStudio project named jsPsychMaker-0.0.2 will open.

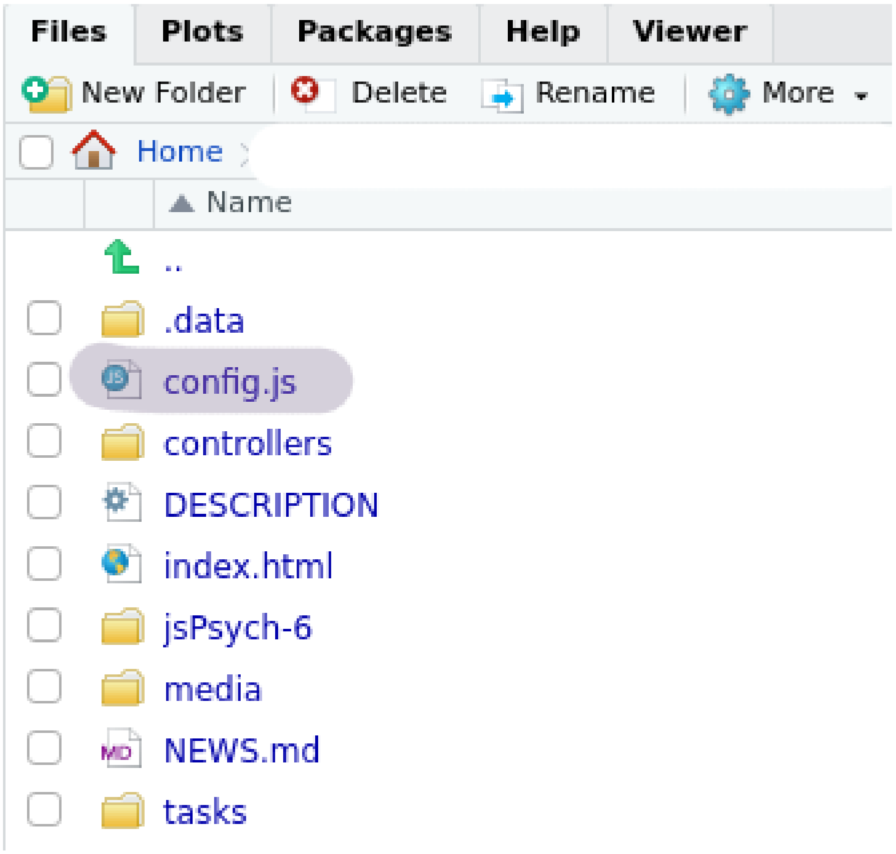
```
# Make sure you have {usethis} installed
if (!require('usethis')) install.packages('usethis'); library('usethis')

# Download last stable jsPsychMaker version to the Downloads folder
usethis::use_course(url = "https://github.com/gorkang/jsPsychMaker/archive/refs/tags/v

# Alternatively, download the dev version:
# usethis::use_course(url = "gorkang/jsPsychMaker", destdir = "~/Downloads/")
```

2) Edit configuration

Go to folder `canonical_protocols` and edit `config.js`. You can also use the jsPsychMaker_config Shiny APP and copy the generated config.js file to your protocol folder.



The variable `tasks` should contain the name of an array of the tasks we want to run.

Available tasks:
- full list of tasks in the Github repo
- details of available tasks (Spanish)
- If you need a NEW task fill this form

3) Run experiment

The experiment is ready to run on your computer. Open `index.html` in Google Chrome or your favorite (and up to date) browser.

3.2 jsPsychMonkeys: Simulate participants

See the jsPsychMonkeys chapter for more detailed instructions.

---

jsPsychMonkeys uses Selenium inside a Docker container to guarantee each session is a clean session. On Linux it's use and configuration is trivial, but on Windows it can be trickier.

Outline

1) Download jsPsychMonkeys

2) Setup

3) Run Monkeys

---

1) Download jsPsychMonkeys

```
if (!require('usethis')) install.packages('usethis'); library('usethis')

usethis::use_course(url = "gorkang/jsPsychMonkeys", destdir = "~/Downloads/")
```

2) Setup

Run setup: `source("setup.R")`. This will install the packages needed.

Ubuntu

You may need to install some system libraries first:

- `sudo apt install libssl-dev libcurl4-openssl-dev libxml2-dev`

Mac

- (?)

Windows

Download and install docker:

- https://docs.docker.com/docker-for-windows/install/ (~ 500MB)

3) Run Monkeys

Open `run.R` and follow the instructions. Mainly:

- Open `_targets.R` file: `rstudioapi::navigateToFile("_targets.R")`
- Edit `parameters_monkeys_minimal`. For example, launch participants 1 to 5 to the 999 protocol locally:
  - `parameters_monkeys_minimal = list(uid = 1:5, local_folder_tasks = "Downloads/999")`
- Run Monkeys!: `targets::tar_make()`

3.3 jsPsychHelpeR: Prepare data

See the jsPsychHelpeR chapter for more detailed instructions.

Outline

1) Download jsPsychHelpeR

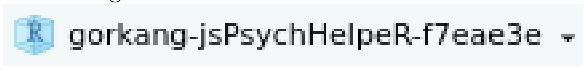
2) Run setup

3) Run data preparation

---

1) Download jsPsychHelpeR

- Run the following two lines in a RStudio console. A new RStudio session will appear. On the top right corner of your RStudio you should see something similar to:



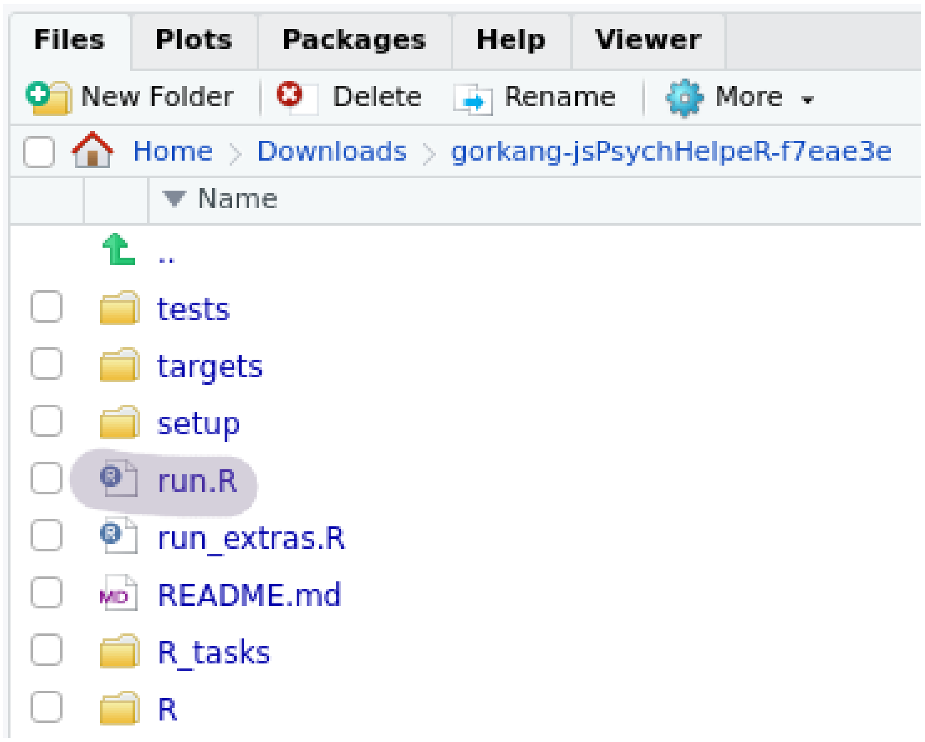
gorkang-jsPsychHelpeR-f7eae3e

```r
# Make sure you have {usethis} installed
if (!require('usethis')) install.packages('usethis'); library('usethis')

# Download jsPsychHelpeR version to the Downloads folder
usethis::use_course(url = "gorkang/jsPsychHelpeR", destdir = "~/Downloads/")
```

2) Run setup

Open the file `run.R` and run initial setup:



```r
# Load functions
invisible(lapply(list.files("./R", full.names = TRUE, pattern = ".R$"), source))

# Run initial setup [REMEMBER to replace `999` with your project number]
run_initial_setup(pid = 999)
```

`run_initial_setup()` will:

- Delete old files
- Install dependencies
- Create necessary folders
- Download results files for the project (you will need the FTP credentials)
- Download and zip a copy of the full protocol without the data (you will need the FTP credentials)
- Create a `_targets.R` file for your project

IMPORTANT: You may need to manually copy the results files to the folder `data/PROJECT ID`

3) Run data preparation

```
# Run data preparation
targets::tar_make()
```

If you are curious, running `targets::tar_visnetwork(targets_only = TRUE)` will show the whole data preparation targets tree.

Chapter 4

jsPsychMaker

---

jsPsychMaker: Create experiments with jsPsych, randomize participants, etc.

Using jsPsychMaker to build experimental protocols helps you with a few things:

- Create full protocols using tasks already implemented by just editing a config.js file
  - You can also use the jsPsychMaker Shiny APP to create your config.js file
- Select order of tasks, randomize subsets of tasks, etc.
- Randomize participants to groups making sure the balance between the groups is maintained
- Allow participants to continue in the task where they left in the protocol
- Set time limits to complete the protocol
  - Automatically discard participants over the time limit, freeing the slots for new participants
- Seamlessly select between online and offline protocols

- Simulate participants with jsPsychMonkeys

- *Automagically* get your data prepared with jsPsychHelpeR

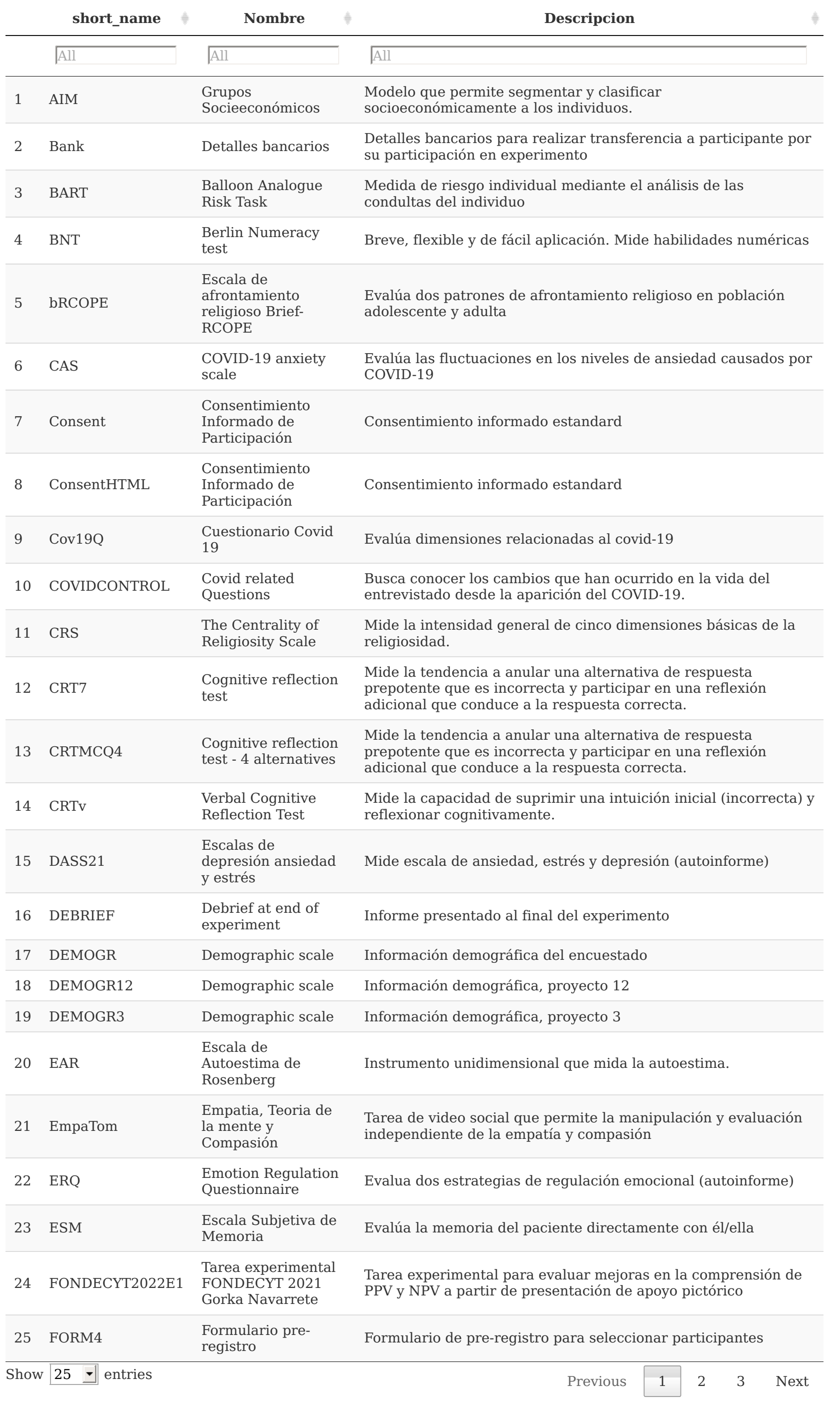
See QuickGuide for basic instructions.

4.1 Available tasks

In 2022-07-31 we have 65 tasks implemented, and 28 in development. The full details about the available tasks can be checked in this document. You can always check the full list of tasks in the Github repo.

If you need help creating a NEW task fill this form.

Below, a table with an overview of the available tasks:



	short_name	Nombre	Descripcion
	All	All	All
1	AIM	Grupos Socieeconómicos	Modelo que permite segmentar y clasificar socioeconómicamente a los individuos.
2	Bank	Detalles bancarios	Detalles bancarios para realizar transferencia a participante por su participación en experimento
3	BART	Balloon Analogue Risk Task	Medida de riesgo individual mediante el análisis de las condultas del individuo
4	BNT	Berlin Numeracy test	Breve, flexible y de fácil aplicación. Mide habilidades numéricas
5	bRCOPE	Escala de afrontamiento religioso Brief-RCOPE	Evalúa dos patrones de afrontamiento religioso en población adolescente y adulta
6	CAS	COVID-19 anxiety scale	Evalúa las fluctuaciones en los niveles de ansiedad causados por COVID-19
7	Consent	Consentimiento Informado de Participación	Consentimiento informado estandard
8	ConsentHTML	Consentimiento Informado de Participación	Consentimiento informado estandard
9	Cov19Q	Cuestionario Covid 19	Evalúa dimensiones relacionadas al covid-19
10	COVIDCONTROL	Covid related Questions	Busca conocer los cambios que han ocurrido en la vida del entrevistado desde la aparición del COVID-19.
11	CRS	The Centrality of Religiosity Scale	Mide la intensidad general de cinco dimensiones básicas de la religiosidad.
12	CRT7	Cognitive reflection test	Mide la tendencia a anular una alternativa de respuesta prepotente que es incorrecta y participar en una reflexión adicional que conduce a la respuesta correcta.
13	CRTMCQ4	Cognitive reflection test - 4 alternatives	Mide la tendencia a anular una alternativa de respuesta prepotente que es incorrecta y participar en una reflexión adicional que conduce a la respuesta correcta.
14	CRTv	Verbal Cognitive Reflection Test	Mide la capacidad de suprimir una intuición inicial (incorrecta) y reflexionar cognitivamente.
15	DASS21	Escalas de depresión ansiedad y estrés	Mide escala de ansiedad, estrés y depresión (autoinforme)
16	DEBRIEF	Debrief at end of experiment	Informe presentado al final del experimento
17	DEMOGR	Demographic scale	Información demográfica del encuestado
18	DEMOGR12	Demographic scale	Información demográfica, proyecto 12
19	DEMOGR3	Demographic scale	Información demográfica, proyecto 3
20	EAR	Escala de Autoestima de Rosenberg	Instrumento unidimensional que mida la autoestima.
21	EmpaTom	Empatia, Teoria de la mente y Compasión	Tarea de video social que permite la manipulación y evaluación independiente de la empatía y compasión
22	ERQ	Emotion Regulation Questionnaire	Evalua dos estrategias de regulación emocional (autoinforme)
23	ESM	Escala Subjetiva de Memoria	Evalúa la memoria del paciente directamente con él/ella
24	FONDECYT2022E1	Tarea experimental FONDECYT 2021 Gorka Navarrete	Tarea experimental para evaluar mejoras en la comprensión de PPV y NPV a partir de presentación de apoyo pictórico
25	FORM4	Formulario pre-registro	Formulario de pre-registro para seleccionar participantes

Show 25 entries

Previous 1 2 3 Next

4.2 Experiment configuration

In the `config.js` file you can find the main parameters to control how your experiment works.

There is a shiny app in `jsPsychMaker/app/` to help you create a `config.js` with the main parameters for your protocol using a visual interface. You can run the app locally or go to jsPsychMaker_config Shiny APP. You will need to copy the generated config.js file to your protocol folder. The Shiny app can also help you create a parametrized consent form (see the Consent tab).

4.2.1 Main parameters

- `pid = 999999;`: Number of protocol
- `online = true;`: true if the protocol run in a server, false if runs locally
- `max_participants = 3;`: If you have `between participants` conditions (participants are assign to only one of a number of conditions), this is the max number of participants per condition
- `random_id = false;`: true if you want to assign a random id to participants, false if the participant needs to input an id
- `max_time = "24:00:00";`: Max time to complete the protocol (HH:MM:SS; Hours:Minutes:Seconds)
- `accept_discarded = true;`: If an user is discarded (over max_time), shall be allowed to continue given there are available slots?
- `debug_mode = false;`: When testing the protocol it shows DEBUG messages, creates the DB tables if they don't exist… It also forces the tasks with random order of items to avoid the randomization so the `jsPsychMonkeys` can have a reproducible behavior.

4.2.2 Order of tasks

- `first_tasks = ['Consent'];`: The protocol will start with these tasks in the specified order
- `last_tasks = ['Goodbye'];`: These tasks will be presented in the specified order after `randomly_ordered_tasks`

Create as many blocks as needed:

- `randomly_ordered_tasks_1 = ['TASK1', 'TASK2'];`: Block of tasks in random order

- `randomly_ordered_tasks_2 = ['TASK3'];`: Block of tasks in random order
- `secuentially_ordered_tasks_1 = ['TASK5', 'TASK4'];` // Block of tasks in sequential order

The final array of tasks can be build combining the above blocks. The order of the tasks in the arrays starting with "random" will be randomized.

- `tasks = ['first_tasks', 'randomly_ordered_tasks_1', 'secuentially_ordered_tasks_1', 'randomly_ordered_tasks_2', 'last_tasks']`

4.2.3 Between-subject tasks

The variable `all_conditions` in `config.js` let's you define the Independent Variables (IV) and levels for the between-subject tasks:

- If there is no between-subject task:
  `all_conditions = {"protocol": {"type": ["survey"]}};`
- If there are between-subject tasks: `all_conditions = {"NAMETASK": {"name_IV": ["name_level1", "name_level2"]}};`

`jsPsychR` will randomize participants to the different conditions keeping the unbalance between conditions to the minimum possible.

4.3 online-offline protocols

jsPsych uses standard web technologies (HTML, CSS y Javascript), so that protocols should run in any modern browser (updated, please). We recommend Google Chrome just because our test suite runs with Google Chrome, so we will catch its specific issues earlier.

4.3.1 Offline

If you want to run a protocol locally (on your computer, on a lab computer), you need to:

- set `online = false;` in the `config.js` file
- double click index.html

`jsPsychR` will use `IndexedDB` to store the participants' progress and balance between conditions. The output csv files will be Downloaded to the Download folder of the computer where the protocol runs.

4.3.1.1 CORS ERRORS

If any of the tasks imports an html file, the Offline protocol will give a CORS error.

There are ways to disable web security in your browser, but it **MUST only be done if your experiment computer runs offline**, otherwise you will be exposed to very bad things.

See how to run chrome disabling web security to avoid CORS error: https://stackoverflow.com/questions/3102819/disable-same-origin-policy-in-chrome)

- google-chrome –disable-web-security –user-data-dir="~/"

4.3.2 Online

Tu run a protocol online, set `online = true;` in the `config.js` file. You will need a couple more things:

- MySQL running in your server
- A file `.secrets_mysql.php` with the content below
- Define the route to `.secrets_mysql.php` in `controllers/php/mysql.php`
  - `require_once '../../.secrets_mysql.php';`
  - THIS FILE **MUST NOT** BE PUBLICLY VISIBLE FROM THE BROWSER
- **Upload the files to the server** :)

```
<?php

/* DO NOT UPLOAD TO PUBLIC REPO */

  $servername = "127.0.0.1";
  $username = "USERNAME OF THE DATABASE";
  $password = "PASSWORD OF THE DATABASE";
  $dbname = "NAME OF THE DB";

?>
```

`jsPsychR` will use `MySQL` to store the participants' progress and balance between conditions. The output csv files will be Downloaded in the `.data/` folder inside the protocol folder in the server.

Before launching the final experiment, make sure you start with a clean slate! That can be summarized in 3 simple steps:

1. Check the configuration for you experiment (`config.js`) and make sure all is well. Some of the critical bits are:

```
pid = 999; // SHOULD have your project ID!
online = true; // true is good
max_participants = 100; // Max participants per contition [number]
max_time = "24:00:00"; // Max time to complete the protocol [HH:MM:SS]
debug_mode = false; // SHOULD be false
```

2. Check that the `.data/` folder for your protocol is empty in the server. You will likely have remains of the piloting and Monkeys.

3. Clean up the MySQL data associated to your protocol.

```
SET @PID = 999; // HERE YOUR PROTOCOL ID!

delete from experimental_condition where id_protocol=@PID;
delete from user where id_protocol=@PID;
delete from user_condition where id_protocol=@PID;
delete from user_task where id_protocol=@PID;
delete from task where id_protocol=@PID;
delete from protocol where id_protocol=@PID;
```

You will most likely need help from the server admin to perform these steps.

4.4 Developing tasks

Remember to place an `if (debug_mode == 'false')` before the randomization of the item order so when running in debug_mode, the items are not randomized. This is important so the behaviour of the jsPsychMonkeys is reproducible:

```
if (debug_mode == 'false') NAMETASK = jsPsych.randomization.repeat(NAMETASK,1);
```

4.4.1 Need help implementing a task!

If you need help developing new tasks, you can open a new Issue in the jsPsychMaker Github.

We will ask you to add the details about the task in the NEW tasks document.

Once the task is implemented, our goal is to always end up having a sister task preparation script in jsPsychHelpeR. You can try to create the preparation script and do a Pull request, or ask for help opening a new Issue in the jsPsychHelpeR Github.

4.5 Technical aspects

When index.html is launched:

- Checks if there are available slots

When an uid is assigned:

- `questions` array is created
- `between-participants` conditions are assigned and stored in the DB (MySQL if online, IndexedDB if offline)

Each question, timeline or conditional question needs to have a:

`data: {trialid: 'NameTask_001', procedure: 'NameTask'}`

The `trialid` identifies the trial, and the `procedure` makes possible to find that trial so participants can continue the tasks where they left, know when participants finished the tasks, etc. This is done in MySQL if online, IndexedDB if offline.

`trialid`'s need to have a standardized structure, which generally conforms with `NameTask_3DigitNumber`. When using conditional items the structure can be a bit more complex, but not much. We use the following rules to check for non-complying trialid's:

- [1]{1,100}_[0-9]{2,3}$ -> `NameTask_2or3DigitNumber`, for example `BNT_001`

- [2]{1,100}*[0-9]{2,3}*[0-9]{1,3}$ -> `NameTask_2or3DigitNumber_1to3DigitsSuffix`, for example `BNT_002_1`

- [3]{1,100}_[0-9]{2,3}_if$ -> `NameTask_2or3DigitNumber`, for example `BNT_002_if`

- [4]{1,100}*[0-9]{2,3}*[0-9]{1,3}_if$ -> `NameTask_2or3DigitNumber`, for ex-

[1]a-zA-Z0-9
[2]a-zA-Z0-9
[3]a-zA-Z0-9
[4]a-zA-Z0-9

ample `BNT_002_1_if`

4.5.1 jsPsychMaker main changes on a task

1. Start of a task

```
questions = ( typeof questions != 'undefined' && questions instanceof Array ) ? questions : [];
questions.push( check_fullscreen('NameOfTask') );
NameOfTask = [];
```

2. Each item

```
data: {trialid: 'NameOfTask_01', procedure: 'NameOfTask'}
```

3. End of experiment

```
if (debug_mode == 'false') NameOfTask = jsPsych.randomization.repeat(NameOfTask, 1);
NameOfTask.unshift(instruction_screen_experiment);
questions.push.apply(questions, NameOfTask)

questions.push({
    type: 'call-function',
    data: {trialid: 'NameOfTask_000', procedure: 'NameOfTask'},
    func: function(){
      if (online) {
        var data = jsPsych.data.get().filter({procedure: 'NameOfTask'}).csv();
      } else {
        var data = jsPsych.data.get().filter({procedure: 'NameOfTask'}).json();
      }
      saveData(data, online, 'NameOfTask');
    }
});
```

4.5.2 Conditional questions

```
var question001 = {
  type: 'survey-multi-choice-vertical',
  questions: [{prompt: '<div class="justified">¿Usted se ha vacunado contra el coronavirus / covi
  data: {trialid: 'PVC_001', procedure: 'PVC'}
};
PVC.push(question001);

var question001_1 = {
  type: 'survey-multi-choice-vertical',
  questions: [{prompt: '<div class="justified">¿Usted se va a vacunar contra el coronav
  data: {trialid: 'PVC_001_1', procedure: 'PVC'}
};

var if_question001_1 = {
  timeline: [question001_1],
  data: {trialid: 'PVC_001_1_if', procedure: 'PVC'},
  conditional_function: function(){
    let data = (JSON.parse((jsPsych.data.get().values().find(x => x.trialid === 'PVC_00
    if((data) ==  'No'){
      return true;
    } else {
      return false;
    }
  }
};
PVC.push(if_question001_1);
```

4.6 Common ERRORS

If you get the following error in the console: `Uncaught TypeError: Cannot read properties of undefined (reading 'procedure')`

Run this in the console:

```
for (var i = 0; i < questions.length; i++) {
  console.log(i + questions[i].data["procedure"])
}
```

It will stop in one of the items. Go to the console, check the array `questions` and go to the number that failed.

When you know the task and item that fails, you probably need to add:

- `data: {trialid: 'TASKNAME_ITEMNUMBER', procedure: 'TASKNAME'}`

Chapter 5

jsPsychMonkeys

---

jsPsychMonkeys: Release monkeys to a jsPsych experiment using the R package {targets}, docker and {RSelenium}.

With jsPsychMonkeys you can simulate participants and easily do the following:

- Simulate participants online and offline
- Simulate participants sequentially and in parallel
- Ask your Monkeys to take pictures of each screen
- Make the behavior of the Monkeys reproducible setting a random seed associated with their unique id
- Store logs of the process, including console logs with errors
- Watch your participants randomly click things in VNC

See QuickGuide for basic instructions.

5.1 How to simulate participants

If the QuickGuide configuration steps didn't work… You may need to do one of the things below:

- Run participants manually
- Use a Linux computer or create a Linux partition
- Create a Linux virtual machine from where to simulate participants. You can use Virtualbox to install Ubuntu. Once there, you can use this manual to prepare the system to run R & RStudio

5.2 Parameters

Edit the `Parameters` section of the `_targets.R` file. The minimal set of parameters needed are:

```
parameters_monkeys_minimal = list(
  uid = 1:10, # User id's for the participants.
  local_folder_tasks = "Downloads/tests/2" # Location of your jsPsych protocol
)
```

5.2.1 Release the Monkeys!

If you want a sequential process:

- `targets::tar_make()`

If you want a parallel horde of monkeys:

- `targets::tar_future_make(workers = 2)`

You can set as many parallel workers as you want. With `targets::tar_make_future(workers = future::availableCores() - 1)` you can have as many workers as your computer cores minus 1.

In `run.R` you can see the typical set of commands needed for a variety of situations.

---

10 Monkeys completing a protocol in parallel. You can use `targets::tar_watch(seconds = 10, outdated = FALSE, targets_only = TRUE)` to see the live progress:

5.2.2 Launch monkeys on a server

You will need a `.vault/SERVER_PATH.R` file that contains the path where the protocols are located in your server: `server_path = "http://URL_OF_YOUR_SERVER/PROTOCOLS_GENERAL_FOLDER/"`

With the `server_folder_tasks` you will set the subfolder where the protocol is located. In the example below the monkeys would go to, `http://URL_OF_YOUR_SERVER/PROTOCOLS_GENERAL_FOLDER/1`

```
parameters_monkeys_minimal = list(
  uid = 1:10, # User id's for the participants.
  server_folder_tasks = "1" # Location of your jsPsych protocol
)
```

5.2.3 Parameters for parameters_monkeys_minimal

There are a few parameters that can be useful:

- `uid_URL = TRUE`: The uid is passed in the URL (e.g. `&uid=1`)
- `local_folder_tasks = rep("Downloads/tests/test_prototol", 25)`: Passing a vector of multiple tasks will make the monkeys to complete all of them.
- `keep_alive = TRUE` Keep the docker container alive after completing the tasks
- `DEBUG = TRUE` Activate DEBUG mode. Lot's of stuff will show up in the console.
- `open_VNC = TRUE` Activate DEBUG mode and open a VNC container to see the monkey's progress.
- `screenshot = TRUE` The monkeys will take a picture of all the pages they see. The .png files are stored in `outputs/screenshots`
- `debug_file = TRUE` Activate DEBUG mode and store all the console output in the `outputs/log`
- `big_container = TRUE` Sets the Shared memory size (/dev/shm) to 2 gigabytes. This is useful to avoid long/complex protocols to crash
- `disable_web_security = TRUE` If you are running a local protocol that loads external files (e.g. consent form in a html file), you may need this. Only works with Google Chrome.
- `console_logs = TRUE` Store the browser's console logs. Only works with Google Chrome

- `forced_random_wait = TRUE` Will wait a randomly sampled number of seconds on page 4
- `forced_seed = 11` Set a random seed so the Monkeys' behavior will be fully reproducible
- `forced_refresh = 20` Refresh browser in page 20 (if TRUE is given, it will refresh in a randomly sampled page)

5.2.3.1 Parameters details

- `local_folder_tasks`: If the folder is not accessible to Docker (anything outside the Download folder), jsPsychMonkeys will create a copy of the protocol in `Downloads/JSPSYCH/`

Chapter 6

jsPsychHelpeR

---

jsPsychHelpeR: Standardize and automatize data preparation and analysis of jsPsych experiments created with jsPsychMaker.

---

jsPsychHelpeR will lend you a hand automatizing and standardizing your data preparation and analysis.

- Use a completely open, reproducible and automatic process to prepare your data
- Data preparation ready for > 50 tasks (see here the list of tasks)
- Get tidy output dataframes for each task, and tidy general dataframes for the whole protocol
- Include tests for common issues
- Automatic reports with progress, descriptives, codebook, etc.

See QuickGuide for basic instructions.

6.1 How to prepare data

Our goal is that jsPsychMaker task has a sister script on jsPsychHelpeR to help prepare the data automatically. If a task you need does not have one, you can try to create the script yourself and do a pull request in the jsPsychHelpeR repo, or fill the NEW tasks document with the details to help us create the correction script.

If you already ran a pilot experiment, simply:

1. Download `jsPsychHelpeR`:

```
if (!require('usethis')) install.packages('usethis'); library('usethis')
usethis::use_course("gorkang/jsPsychHelpeR")
```

2. `run_initial_setup()`: go to the file `run.R` and run the following two lines. They will:

- Try to make sure you have all the dependencies, folders, etc.
- Download all the data from your protocol (you will need the FTP credentials and set `download_files = TRUE`)
- Download and zip a copy of the full protocol without the data (you will need the FTP credentials and set `download_task_script = TRUE`)
- Create a customized `_targets.R` file adapted to the data of your protocol, so data preparation can run automagically

```
invisible(lapply(list.files("./R", full.names = TRUE, pattern = ".R$"), source))
run_initial_setup(pid = "999", download_files = TRUE, download_task_script = TRUE)
```

This should work on Ubuntu, if you have the FTP credentials, and `sshpass` and `rsync` installed.

If you don't have the credentials, or some of the requirements to download the files, you can manually copy the .csv files to the `data/PROJECT ID` folder, and then run the `offline` version of `run_initial_setup()`:

```
invisible(lapply(list.files("./R", full.names = TRUE, pattern = ".R$"), source))
run_initial_setup(pid = "999", download_files = FALSE, download_task_script = FALSI
```

We use the targets package.

The whole process can be reproduced running `targets::tar_make()`

A nice visualization of all the pre-processing steps can be seen with `targets::tar_visnetwork(targets_only = TRUE)`

The file `_targets.R` contains the important parameters and calls to all the functions used when running `targets::tar_make()`

To see more detail about any specific step, you can:

1. Go to the relevant function
2. Load the input parameters of the function with `debug_function(NAME_OF FUNCTION)`. Alternatively, manually use `targets::tar_load(NAME_OF_TARGET)`
3. Run the code step by step as you would normally do

6.2 Basics

jsPsychHelpeR uses as input a data created with a jsPsychMaker experimental protocol.

6.2.1 Inputs

The input data folder will be named after the protocol_id, for example `999/` and needs to be placed in the `data/` folder of the jsPsychHelpeR project `data/YOUR_PROJECT_NUMBER`:

- The data folder can contain either multiple .csv files, or a single .zip file

There will be a single .csv file for each participant and task of the protocol. For example:

- 999_Consent_original_2022-04-02T205622_1.csv:
  - [project: 999]_[experimento: Consent]_[version: original]_[datetime: 2022-04-02T205622]_[participant id: 1]

6.2.2 Outputs

When the pipeline successfully runs with `targets::tar_make()`, a number of outputs will be created.

All the outputs can be found in the `/outputs` folder. The only exception is the sensitive data and reports, which can be found in `.vault/outputs`. WARNING: The '.vault/' folder **MUST NOT** be made public.

6.2.2.1 Output folders

The outputs will be organized in different folders:

- **Data frames** for different stages of data processing can be found in `outputs/data`
- **Temporary files for manual correction** are in `outputs/data/manual_correction` (the final manual correction files must be place by the user in `data/manual_correction`). WARNING: These will be overwritten each time the pipeline runs
- **Plots**, **tables** and **reports** are in `outputs/plots`, `outputs/tables`and `outputs/reports` respectively.
- **Test outputs** are in `outputs/tests_outputs`
- **Anonymized Raw data** will be moved to `.vault/data_vault/`

6.2.2.2 Output dataframes

There will be a single data frame (df) for each of the tasks in `outputs/data`, plus a data frame (DF) for each of the steps of the data preparation, and a dictionary file listing all the available tasks. We store the files in two formats, csv and rds:

- **DF_raw.csv**: All the `data/project_id/` csv files combined on a single file. We only add the columns "project", "experimento", "version", "datetime", "id" by parsing the filenames
- **DF_clean.csv**: Clean version of the raw file ready to process the individual tasks
- **df_ShortNameOfTask.csv**: One df for each of the tasks of the protocol after being processed with the `prepare_ShortNameOfTask()` functions
- **DF_joined.csv**: all the processed tasks joined in a single DF
- **DF_analysis**: only the total scores and dimensions from `DF_joined` (columns ending in `_DIRt`, `_STDt`, `_DIRd`, `_RELd`, `STDd`). Can be visually explored using the shiny app in `Rmd/app.R`
- **DICCIONARY_tasks.csv**: list of all tasks in the protocol

6.2.2.3 Output dataframes column names

All the output processed data frames columns are named in a standardized way:

- **ShortNameOfTask_ItemNumber_RAW**: raw responses of participants for individual items
- **ShortNameOfTask_ItemNumber_DIR**: processed raw responses following the task correction instructions (e.g. inverting certain items, converting strings to numbers, computing accuracy…)
- **ShortNameOfTask_RAW_NA**: number of missing data (NA) in the RAW responses
- **ShortNameOfTask_DIR_NA**: number of missing data (NA) in the DIR responses. If it is not equal to `ShortNameOfTask_RAW_NA` there is something wrong in the items correction.
- **ShortNameOfTask_DimensionName_DIRd**: scores for a specific dimension (`d`) in a task, calculated following task correction instructions (e.g. summing or averaging certain items)
- **ShortNameOfTask_DimensionName_RELd**: scores for a specific dimension (`d`) in a task, calculated following task correction instructions AND after filtering items with low reliability. See Reliability section for more information.
- **ShortNameOfTask_DimensionName_STDd**: standardized score for a dimension (`d`)
- **ShortNameOfTask_DIRt**: total (`t`) score for a task calculated following task correction instructions (e.g. summing or averaging all items)
- **ShortNameOfTask_STDt**: standardized (`t`) score for a task

6.3 Advanced

6.3.1 Create your own reports

You can use any of the template reports in the `_targets.R` file, or create your own reports.

We will start opening one of the template reports: `rstudioapi::navigateToFile("doc/report_analysis.Rmd")`.

- Edit the RMarkdown file to adapt it to your needs.
- If you already did `targets::tar_make()`, when running `targets::tar_load(DF_analysis)` the dataframe `DF_analysis` will load in your Environment.

Go back to the `_targets.R` file:

- Look for `# Analysis report` and uncomment the following lines:

```
# tar_render(report_analysis, "doc/report_analysis.Rmd",
#            output_file = paste0("../outputs/reports/report_analysis.html")),
```

When you finished editing and uncomented the tar_render command, go back to the `run.R` file:

- `targets::tar_make()`

6.3.2 Create new tasks

To create the correction script for a new task, you start with:

- `create_new_task(short_name_task = "NAMETASK")`

This will:

- create a new file from a template correction script (`R_tasks/prepare_TEMPLATE.R`)
- adapt it to your `short_name_task` to make everything as standardized as possible
- open the new `prepare_NAMETASK.R` file

If the parameter `get_dimensions_googledoc = TRUE`:

- The NEW tasks document is checked.
- If the document has been filled properly, it will show in the console standardized strings (ready to be copy/pasted to the new `prepare_NAMETASK.R` file) about:
  - dimension names
  - items corresponding to each dimension
  - dimension calculation
  - inverse items
  - numeric conversion of items

You can also use `get_dimensions_googledoc()` as a standalone function:

`get_dimensions_googledoc(short_name_text = "MLQ")`

All the `prepare_NAMEOFTASK.R` scripts on the `R_tasks/` folder have been created starting from the same template. The only exception are the experimental tasks and some surveys with particularities that require more complex adaptations.

When you finish implementing the correction script, please do a Pull request so we can add you script to the pool. If you have not already, please help us filling up details about the task in the NEW tasks document.

6.3.3 Adapting new tasks

`get_dimensions_googledoc` will show you how to adapt the `prepare_TASK()` script, but you will need to know how it works to be able to edit the relevant bits. Also, sometimes `get_dimensions_googledoc` won't get all the details of the task right, or there could be non-standard elements to it. Here, we will describe some of the elements of the template to help understand how it works.

Remember you should **ALWAYS** start with `create_new_task(short_name_task = "NAMETASK")` so your task template works well with `jsPsychHelpeR`.

There are three chunks you will need to adapt to have a fully working preparation script.

- `[ADAPT 1/3]: Items to ignore and reverse, dimensions`
- `[ADAPT 2/3]: RAW to DIR for individual items`
- `[ADAPT 3/3]: Scales and dimensions calculations`

6.3.3.1 Items to ignore and reverse, dimensions

```
# [ADAPT 1/3]: Items to ignore and reverse, dimensions -----------------------
# ***************************************************************************

description_task = "" # Brief description here

items_to_ignore = c("000") # Ignore these items: If nothing to ignore, keep as is
items_to_reverse = c("000") # Reverse these items: If nothing to reverse, keep as is

## NameDimension1, NameDimension2 should be the names of the dimensions
## Inside each c() create a vector of the item numbers for the dimension
## Add lines as needed. If there are no dimensions, keep as is
items_dimensions = list(
  NameDimension1 = c("000"),
  NameDimension2 = c("000")
)

# [END ADAPT 1/3]: ***********************************************************
# ****************************************************************************
```

6.3.3.2 RAW to DIR for individual items

```
DF_long_DIR =
   DF_long_RAW %>%
   select(id, trialid, RAW) %>%



 # [ADAPT 2/3]: RAW to DIR for individual items -------------------------------
 # ****************************************************************************

   # Transformations
   mutate(
     DIR =
       case_when(
         RAW == "Nunca" ~ 1,
         RAW == "Poco" ~ 2,
         RAW == "Medianamente" ~ 3,
         RAW == "Bastante" ~ 4,
         RAW == "Mucho" ~ 5,
         is.na(RAW) ~ NA_real_,
         grepl(items_to_ignore, trialid) ~ NA_real_,
         TRUE ~ 9999
       )
   ) %>%

   # Invert items
   mutate(
     DIR =
       case_when(
         DIR == 9999 ~ DIR, # To keep the missing values unchanged
         trialid %in% paste0(short_name_scale_str, "_", items_to_reverse) ~ (6 - DIR)
         TRUE ~ DIR
       )
   )

 # [END ADAPT 2/3]: ***********************************************************
 # ****************************************************************************
```

6.3.3.3 Scales and dimensions calculations

```
# [ADAPT 3/3]: Scales and dimensions calculations ---------------------------
# ***************************************************************************

# Reliability -------------------------------------------------------------
# REL1 = auto_reliability(DF_wide_RAW, short_name_scale = short_name_scale_str, items = items_D
# items_RELd1 = REL1$item_selection_string


# [USE STANDARD NAMES FOR Scales and dimensions: names_list$name_DIRd[1], names_list$name_DIRt,
# CHECK with: create_formulas(type = "dimensions_DIR", functions = "sum", names_dimensions)
DF_wide_RAW_DIR =
  DF_wide_RAW %>%
  mutate(

    # [CHECK] Using correct formula? rowMeans() / rowSums()

    # Score Dimensions (see standardized_names(help_names = TRUE) for instructions)
    !!names_list$name_DIRd[1] := rowMeans(select(., paste0(short_name_scale_str, "_", items_dim
    !!names_list$name_DIRd[2] := rowSums(select(., paste0(short_name_scale_str, "_", items_dime

    # Reliability Dimensions (see standardized_names(help_names = TRUE) for instructions)
    # !!names_list$name_RELd[1] := rowMeans(select(., paste0(short_name_scale_str, "_", items_R

    # Score Scale
    !!names_list$name_DIRt := rowSums(select(., matches("_DIR$")), na.rm = TRUE)

  )

# [END ADAPT 3/3]: ***********************************************************
# ***************************************************************************
```

6.3.4 How to fill the NEW tasks document

The best way is to check the main document with information about all the tasks (Tareas jsPsychR) and find a similar task to copy/paste and adapt it in the NEW tasks document.

The main suggestion is to be very consistent. For example, when entering the informacion about numeric conversion in the Puntajes_items tab:

All the cells must be:
1 = Mucho
2 = Poco
...

DO NOT do things like:
1: Mucho
1 Mucho
1 pto = Mucho
Mucho 1

Please, make sure you fill out all the details in all the tabs.

6.3.5 DEBUG tasks

At the begining of each of the `R_tasks/prepare_NAMETASK.R` scripts you will find a commented `debug_function(prepare_NAMETASK)` line.

When running it, it will load the input parameters for the task. From there, you can work inside of the preparation scipt as you would normally do in a R script.

If you get the error `"Error in debug_function(prepare_NAMETASK) : could not find function 'debug_function'`debug_function()`does nor work"` you will need to load all the functions in the `R/` folder first.

You can do this in one of three ways:

- `CONTROL + P` shortcut will work if the `run_initial_setup()` completed correctly (at least on Ubuntu systems).
- Run `targets::tar_load_globals()`
- Or directly, source all the scripts in the `R/` folder: `invisible(lapply(list.files("./R", full.names = TRUE, pattern = ".R$"), source))`

6.4 Helper functions

6.4.1 Reliability

You can use the `auto_reliability()` function to help you automatically filter items with low reliability (although doing this automatically is probably a bad idea). The function uses `psych::alpha()` and filters by default items with an r.drop <= 0.2. See `psych::alpha()` help for more details. **IMPORTANT**: Using `psych::omega()` is generally a better idea, see the alpha help page.

An example can be found in prepare_REI40().

The basic logic would be:

```
# Define items for a specific dimension
items_DIRd1 = c("01", "02", "03", "04", "05", "06", "07", "08", "09", "10")

# Calculate reliability
REL1 = auto_reliability(DF_wide_RAW, short_name_scale = short_name_scale_str, items = items_DIRd1

# Store item selection in a variable
items_RELd1 = REL1$item_selection_string

# In the final Dimension calculation, use the item selection including only the items with a reli
## See `items_RELd1` below
!!names_list$name_RELd[1] := rowMeans(select(., paste0(short_name_scale_str, "_", items_RELd1, "_

# Compare it with the calculation including the original items
## See `items_DIRd1` below
!!names_list$name_DIRd[1] := rowMeans(select(., paste0(short_name_scale_str, "_", items_DIRd1, "_
```

6.5 Technical aspects

6.5.1 How trialid's are processed

See `PRFBM`:

- If more than one response per screen
  - Item: `PRFBM_04`
  - Responses: `{"daño":"Parcialmente en desacuerdo","beneficio":"Parcialmente en desacuerdo"}`
  - final trialids: `PRFBM_04_beneficio` and `PRFBM_04_daño`

6.6 Common ERRORS

6.6.1 `run_initial_setup()`:

```
x Can find server credentials in '.vault/.credentials'
x 0 tasks found for protocol 'TU NUMERO DE PROYECTO'. NOT creating _targets.R file
```

6.6.1.1 On Linux (Ubuntu):

- IF you have the server credentials:
  - Open .credentials_TEMPLATE `rstudioapi::navigateToFile(".vault/.credentials_TEM`
  - Edit the file with your server credentials
  - Rename the file to `.credentials`

- IF you DON'T have the credentials but you have the .csv results files:
  - Copy the csv files to the folder `data/YOUR_PROJECT_NUMBER`
  - Run again `run_initial_setup()`

6.6.2 On Mac or Windows:

- Copy the csv files to the folder `data/YOUR_PROJECT_NUMBER`
- Run again `run_initial_setup()`

Chapter 7

jsPsychR Admins

Instructions and protocols for jsPsychR admins. The goal is to minimize issues and make sure tasks behave in a consistent and reproducible manner.

7.1 Google Docs

- All tasks
- List of protocols
- NEW tasks
- Checks specific tasks

7.2 Folders and how to work

We have two main locations, the Github jsPsychMaker project and the server.

Github jsPsychMaker project

- canonical_protocol:
  - machinery: last stable version
  - tasks: all available tasks
  - server: `protocols/999/`
- canonical_protocol_DEV
  - machinery: development version
  - tasks: all available tasks

  - server: `protocols/test/canonical_protocol_DEV/`

- canonical_protocol_clean
  - machinery: last stable version
  - tasks: Consent and Goodbye
  - server: `protocols/test/canonical_protocol_clean/`
- protocols_DEV
  - machinery: last stable version
  - should only contain tasks in development
  - server: `protocols/test/protocols_DEV/`

In `protocols_DEV` we prepare the new **protocolos**:

- Create a copy in `test/protocols_DEV` of `canonical_protocol_clean` and rename to the number of the new protocol, `test/protocols_DEV/NumberOfProtocol`
- Once the protocol is ready:
  - Copy protocol to root folder: `protocols/NumberOfProtocol`
  - ZIP subfolder and move zip to `protocols/test/protocols_DEV/OLD_TESTS/`
  - Delete folder `test/protocols_DEV/NumberOfProtocol`
  - If there are new tasks:
    - CHECK with: check_missing_prepare_TASK()
    - TEST with create_protocol_with_NEW_tasks.R
    - Copy tasks, images, videos, specific plugins, etc. to `protocols/999/`
    - TEST in canonical protocol `protocols/999/` just in case there is a weird interaction

7.3 Helper functions

There are a number of helper functions to make some of the jsPsychR admins tasks easier.

7.3.1 Check all protocols

For example, we can use `check_missing_prepare_TASK()` to:

- Download all the protocols (without data) to a local folder (`sync_protocols = TRUE`)

- Check the trialid's of all the tests are OK (`check_trialids = TRUE`)
- Check there are no duplicate short_name of tasks in the tareas jsPsychR and NUEVAS tareas
- Check which tasks do not have a prepare_TASK.R script
- Check tasks with no info on the tareas jsPsychR Google doc
- Check tasks with missing info on NUEVAS tareas

```
# Open jsPsychHelpeR RStudio project

  # Load check_missing_prepare_TASK() function
  source(here::here("../jsPsychHelpeR/R/check_missing_prepare_TASK.R"))

  # If sync_protocols = TRUE, will download to ../CSCN-server/protocols all the protocols from th
  DF_missing = check_missing_prepare_TASK(sync_protocols = FALSE,
                                          check_trialids = TRUE,
                                          delete_nonexistent = TRUE,
                                          check_new_task_tabs = TRUE)
```

```
## New names:
##
## -- FOLDER: protocols/
## -----------------------------------------------------------
##
## -- Checking /13
## -----------------------------------------------------------------
## * `` -> `...12`

## 1 ISSUES:
##  - experiment: faux_pas
##  - trialid:    faux_pas_001

##
## -- Checking /15 ----------------------------------------------------------------

## 1 ISSUES:
##  - experiment: ITC
##  - trialid:    question_ + pad( (repetition_count * 2)

##
## -- Checking /16 ----------------------------------------------------------------
## 2 ISSUES:
##  - experiment: RMET
##  - trialid:    question000_1, question000_2

##
## -- Checking /18 -----------------------------------------------------------------

## 2 ISSUES:
##  - experiment: RMET
##  - trialid:    question001_1, question001_2

##
## -- Checking /22 -----------------------------------------------------------------

## 2 ISSUES:
##  - experiment: FONDECYT2022E1
##  - trialid:    if_instructions_000, FONDECYT2022E1 + _giro_check_ending

##
## -- Checking /4 ------------------------------------------------------------------

## 6 ISSUES:
##  - experiment: FORM4
##  - trialid:    FORM_01, FORM_02, FORM_03, FORM_04, FORM_05, FORM_06

##
## -- Checking /8 ------------------------------------------------------------------

## 2 ISSUES:
##  - experiment: Goodbye, IRS
##  - trialid:    question001, effort

##
## -- Checking /999 ----------------------------------------------------------------

## 4 ISSUES:
##  - experiment: HRPVB, HRPVBpost
##  - trialid:    HRPVB_BB, HRPVB_MM, HRPVBpost_BB, HRPVBpost_MM

##
## -- FOLDER: protocols/test/protocols_DEV/ ----------------------------------------
##
## -- Checking /22 -----------------------------------------------------------------
## 2 ISSUES:
##  - experiment: FONDECYT2022E1
##  - trialid:    if_instructions_000, FONDECYT2022E1 + _giro_check_ending

##
## -- Checking /NEW_TASKS --------------------------------------------------------

## 8 ISSUES:
##  - experiment: FORM4, RMET
##  - trialid:    FORM_01, FORM_02, FORM_03, FORM_04, FORM_05, FORM_06, question000_1, question00

##
## -- FOLDER: protocols/999/ -----------------------------------------------------
##
## -- Checking /999 --------------------------------------------------------------

## 4 ISSUES:
##  - experiment: HRPVB, HRPVBpost
##  - trialid:    HRPVB_BB, HRPVB_MM, HRPVBpost_BB, HRPVBpost_MM

##
## -- CHECK missing info in tabs -------------------------------------------------
## x Tasks missing info in tabs:
## * altroncoso@alumnosuaicl.onmicrosoft.com: 1
## * buitrago-c@javeriana.edu.co: 10
## * jmoraless@uc.cl: 4
## * joaquin.migeot@gmail.com: 3
## * nicolasmarchant@alumnos.uai.cl: 1
## * vicente.soto.d@uai.cl: 3
##
## -- Details --
##
## * altroncoso@alumnosuaicl.onmicrosoft.com: ESV (citas, puntajes, dimensiones)
## * buitrago-c@javeriana.edu.co: CIT (citas, puntajes, dimensiones); CRQ (citas,
## puntajes, dimensiones); CTT (citas, puntajes, dimensiones); ICvsID (citas,
## puntajes, dimensiones); LSNS (citas, puntajes, dimensiones); MCA (citas,
## puntajes, dimensiones); MDDF (citas, puntajes, dimensiones); PSC (citas,
## puntajes, dimensiones); SCGT (citas, puntajes, dimensiones); UCLA (citas,
## puntajes, dimensiones)
## * jmoraless@uc.cl: CEL (dimensiones); fauxPasEv (citas, puntajes, dimensiones);
## LoB (dimensiones); Words (puntajes, dimensiones)
## * joaquin.migeot@gmail.com: BDI (citas, puntajes, dimensiones); LOT (citas,
## puntajes, dimensiones); MCQ30 (citas, puntajes, dimensiones)
## * nicolasmarchant@alumnos.uai.cl: PPD (citas, puntajes, dimensiones)
## * vicente.soto.d@uai.cl: NARS (citas, puntajes, dimensiones); RMET (citas,
## puntajes, dimensiones); RobToM (citas, puntajes, dimensiones)
##
## -- CHECK duplicates ------------------------------------------------------------
## v No duplicated short_name in the Google docs
```

```
# - Tasks with no prepare_TASK() script!
# - Tasks NOT in Google Doc
# - Check trialid's are OK
DF_missing
```

```
## $DF_FINAL
## # A tibble: 96 x 8
##    task    missing_script missing_gdoc missing_task proto~1 Nombre Descr~2 EMAIL
##    <chr>   <chr>          <chr>        <chr>        <chr>   <chr>  <chr>   <chr>
##  1 AIM     <NA>           <NA>         <NA>         12, 17~ Grupo~ Modelo~ <NA>
##  2 Bank    <NA>           <NA>         <NA>         20, te~ Detal~ Detall~ <NA>
##  3 BART    <NA>           <NA>         <NA>         11, 99~ Ballo~ Medida~ <NA>
##  4 BDI     BDI            <NA>         <NA>         19      <NA>   <NA>    joaq~
##  5 BNT     <NA>           <NA>         <NA>         2, 22,~ Berli~ Breve,~ <NA>
##  6 bRCOPE  <NA>           <NA>         <NA>         999, t~ Escal~ Evalúa~ <NA>
##  7 CAS     <NA>           <NA>         <NA>         8, 999~ COVID~ Evalúa~ <NA>
##  8 CEL     <NA>           <NA>         <NA>         test/p~ Cuest~ Mide l~ jmor~
##  9 CIT     CIT            <NA>         <NA>         23, 28~ Test ~ Prueba~ buit~
## 10 Consent <NA>           <NA>         <NA>         10, 11~ Conse~ Consen~ <NA>
## # ... with 86 more rows, and abbreviated variable names 1: protocols,
## #   2: Descripcion
## # i Use `print(n = ...)` to see more rows
##
## $DF_missing_JS_tasks
## # A tibble: 0 x 3
## # ... with 3 variables: task <chr>, Nombre <chr>, Descripcion <chr>
## # i Use `colnames()` to see all variable names
##
## $DF_missing_script
## # A tibble: 22 x 2
##    task                protocols
##    <chr>               <chr>
##  1 BDI                 19
##  2 CIT                 23, 28, test/protocols_DEV/23, test/protocols_DEV/28
##  3 CRQ                 23, 28, test/protocols_DEV/23, test/protocols_DEV/28
##  4 CTT                 23, 28, test/protocols_DEV/23, test/protocols_DEV/28
##  5 DEMOGRfondecyt2022E1 22, test/protocols_DEV/22
##  6 ESV                 test/protocols_DEV/25
##  7 faux_pas            13
##  8 HC                    test/tasks_DEV
##  9 ICvsID                23, 28, test/protocols_DEV/23, test/protocols_DEV/28
## 10 ITC                   15, 999, test/canonical_protocol_DEV
## # ... with 12 more rows
## # i Use `print(n = ...)` to see more rows
##
## $DF_missing_googledoc
## # A tibble: 5 x 2
##   task                 protocols
##   <chr>                <chr>
## 1 DEMOGR24             test/protocols_DEV/24, test/protocols_DEV/NEW_TASKS
## 2 DEMOGR27             test/protocols_DEV/27, test/protocols_DEV/NEW_TASKS
## 3 DEMOGRfondecyt2022E1 22, test/protocols_DEV/22
## 4 faux_pas             13
## 5 HC                   test/tasks_DEV
##
## $MISSING_tabs
## # A tibble: 6 x 3
##   EMAIL                                 tasks                            TEXT
##   <chr>                                 <chr>                            <chr>
## 1 altroncoso@alumnosuaicl.onmicrosoft.com "ESV (, citas, puntajes, dimens~ altr~
## 2 buitrago-c@javeriana.edu.co           "CIT (, citas, puntajes, dimens~ buit~
## 3 jmoraless@uc.cl                       "CEL (, , , dimensiones, ) ; fa~ jmor~
## 4 joaquin.migeot@gmail.com              "BDI (, citas, puntajes, dimens~ joaq~
## 5 nicolasmarchant@alumnos.uai.cl        "PPD (, citas, puntajes, dimens~ nico~
## 6 vicente.soto.d@uai.cl                 "NARS (, citas, puntajes, dimen~ vice~
```

```r
DF_missing$DF_FINAL %>% tidyr::replace_na(list(missing_script = "",
                                               missing_googledoc = "",
                                               missing_task = ""))
```

```
## # A tibble: 96 x 8
##    task    missing_script missing_gdoc missing_task proto~1 Nombre Descr~2 EMAIL
##    <chr>   <chr>          <chr>        <chr>        <chr>   <chr>  <chr>   <chr>
##  1 AIM     ""             <NA>         ""           12, 17~ Grupo~ Modelo~ <NA>
##  2 Bank    ""             <NA>         ""           20, te~ Detal~ Detall~ <NA>
##  3 BART    ""             <NA>         ""           11, 99~ Ballo~ Medida~ <NA>
##  4 BDI     "BDI"          <NA>         ""           19      <NA>   <NA>    joaq~
##  5 BNT     ""             <NA>         ""           2, 22,~ Berli~ Breve,~ <NA>
##  6 bRCOPE  ""             <NA>         ""           999, t~ Escal~ Evalúa~ <NA>
##  7 CAS     ""             <NA>         ""           8, 999~ COVID~ Evalúa~ <NA>
##  8 CEL     ""             <NA>         ""           test/p~ Cuest~ Mide l~ jmor~
##  9 CIT     "CIT"          <NA>         ""           23, 28~ Test ~ Prueba~ buit~
## 10 Consent ""             <NA>         ""           10, 11~ Conse~ Consen~ <NA>
## # ... with 86 more rows, and abbreviated variable names 1: protocols,
## #   2: Descripcion
## # i Use `print(n = ...)` to see more rows
```

```
# Tasks ready to create prepare_*.R script
DF_missing$DF_FINAL %>% filter(!is.na(missing_script) & is.na(missing_gdoc))
```

```
## # A tibble: 19 x 8
##    task   missing_script missing_gdoc missing_task protoc~1 Nombre Descr~2 EMAIL
##    <chr>  <chr>          <chr>        <chr>        <chr>    <chr>  <chr>   <chr>
##  1 BDI    BDI            <NA>         <NA>         19       <NA>   <NA>    joaq~
##  2 CIT    CIT            <NA>         <NA>         23, 28,~ Test ~ Prueba~ buit~
##  3 CRQ    CRQ            <NA>         <NA>         23, 28,~ Escal~ <NA>    buit~
##  4 CTT    CTT            <NA>         <NA>         23, 28,~ Test ~ <NA>    buit~
##  5 ESV    ESV            <NA>         <NA>         test/pr~ Empat~ <NA>    altr~
##  6 ICvsID ICvsID         <NA>         <NA>         23, 28,~ Impar~ <NA>    buit~
##  7 ITC    ITC            <NA>         <NA>         15, 999~ Inter~ <NA>    <NA>
##  8 LOT    LOT            <NA>         <NA>         19       <NA>   <NA>    joaq~
##  9 LSNS   LSNS           <NA>         <NA>         23, tes~ Escal~ <NA>    buit~
## 10 MCA    MCA            <NA>         <NA>         test/pr~ Moral~ <NA>    buit~
## 11 MCQ30  MCQ30          <NA>         <NA>         19       <NA>   <NA>    joaq~
## 12 MDDF   MDDF           <NA>         <NA>         23, tes~ MORAL~ <NA>    buit~
## 13 NARS   NARS           <NA>         <NA>         18       Negat~ <NA>    vice~
## 14 PPD    PPD            <NA>         <NA>         test/pr~ Proba~ <NA>    nico~
## 15 PSC    PSC            <NA>         <NA>         23, tes~ Proso~ <NA>    buit~
## 16 RobToM RobToM         <NA>         <NA>         18       <NA>   <NA>    vice~
## 17 SCGT   SCGT           <NA>         <NA>         23, tes~ Test ~ Mide l~ buit~
## 18 UCLA   UCLA           <NA>         <NA>         23, tes~ Escal~ <NA>    buit~
## 19 Words  Words          <NA>         <NA>         9        Escal~ Este t~ jmor~
## # ... with abbreviated variable names 1: protocols, 2: Descripcion
```

```
DF_missing$DF_FINAL %>% filter(!is.na(missing_script) | !is.na(missing_gdoc)) %>%
  filter(!task %in% c("DEMOGR24", "DEMOGRfondecyt2022E1", "ITC", "fauxPasEv")) %>%
  select(-matches("missing"), -Nombre, -Descripcion)
```

```
## # A tibble: 21 x 3
##    task     protocols                                                EMAIL
##    <chr>    <chr>                                                    <chr>
##  1 BDI      19                                                       joaquin.migeot~
##  2 CIT      23, 28, test/protocols_DEV/23, test/protocols_DEV/28 buitrago-c@jav~
##  3 CRQ      23, 28, test/protocols_DEV/23, test/protocols_DEV/28 buitrago-c@jav~
##  4 CTT      23, 28, test/protocols_DEV/23, test/protocols_DEV/28 buitrago-c@jav~
##  5 DEMOGR27 test/protocols_DEV/27, test/protocols_DEV/NEW_TASKS  <NA>
##  6 ESV      test/protocols_DEV/25                                    altroncoso@alu~
##  7 faux_pas 13                                                       <NA>
##  8 HC       test/tasks_DEV                                        <NA>
##  9 ICvsID   23, 28, test/protocols_DEV/23, test/protocols_DEV/28 buitrago-c@jav~
## 10 LOT      19                                                    joaquin.migeot~
## # ... with 11 more rows
## # i Use `print(n = ...)` to see more rows
```

7.3.2 Create protocol with NEW tasks

With `create_protocol_with_NEW_tasks.R` we can create a protocol with the tasks for which we do not yet have a control snapshot (no .csv's in the 999.zip data).

This is an important step before the task can go to the canonical protocol.

7.3.3 Check canonical protocol DEV

With `000_CHECK_CANONICAL.R` we can check that the canonical protocol in development works and expected.

In the script you can:

- sync `canonical_protocol_DEV/` folder in jsPsychMaker to `999/` in the server
- launch 5 monkeys
- rename the csv files to a fixed date, etc.
- prepare data
- compare with snaphot (WIP)