neuropsydia Documentation

Release 1.0.4

Dominique Makowski

Aug 14, 2017

Contents

Neuropsydia is a Python module that allows to easily create experiments, tasks or questionnaires.

Contents:

CHAPTER 1

Tutorials

This courses were crafted by psychologists, neuropsychologists and neuroscientists for psychologists, neuropsychologists and neuroscientists. As such, it is a straightforward introduction to Neuropsydia for Python with a special focus on how to actually do something with it. It is not a programming course on Python, nor a course on programming *per se*.

Contents:

1. Getting Started

Contact

For remarks, complaints, suggestions or anything else, please create an issue.

Installation

Installation steps can be found here.

Hands on!

Well, first of all, welcome and thank you for trying the Python version of Neuropsydia for research. I'm sure you will like it as, while still in developpment, it already has some powerful functions that will help you create your tasks, experiments, and more.

But enough talking! After the installation, open your python code editor (*e.g.,* spyder, available within the WinPython bundle). Here, you will write functions that will almost magically come to live once the program is launched.

In spyder, one pane is the **CONSOLE**, where Python actually lives. It's its interface with us humans. If you type something in there, it will swallow it, do something with it, and then, maybe, return something. Also, it's great for calculus. For example, try typing *5*2* and hit Enter.

Pretty cool, huh?

But for now, let's focus on some simple things, such as how to load, start and close neuropsydia.

Import Neuropsydia

Neuropsydia for research is a module which, like any other module, must be loaded in order to be used. In Python, we load modules by running:

```
import themoduleiwanttoload
```

Then, we can use its functions:

```
themoduleiwanttoload.function1()
themoduleiwanttoload.function2()
```

However, as you can see it, it's pretty anoying to write the full name of the module each time we use one of its function. Fortunately for us, Python allows us to load a module under another name (or `alias`), for example the letter "x".

```
import themoduleiwanttoload as x
x.function1()
x.function2()
```

Better. That's what we are going to do with neuropsydia, loading it under the name "n".

However, unlike many other packages, Neuropsydia CANNOT be loaded without two of its function, :code:'start()' and :code:'close()'.

Start and Close

So, when importing `neuropsydia`, what happens is basically that it needs to initialize several things before being ready to use. And those things are initialized with the `start()` function. Moreover, adding the `close()` function at the end will ensure a clean ending.

Therefore, every experiments, after loading the neuropsydia module, will begin by the :code:'start()' function and end with the :code:'close()' function.

Let's try it. Write in the editor the following lines of code:

```
import neuropsydia as n

n.start()
n.close()
```

Execute your code

The rest of your code will go between the `start()` and the `close()` functions. But these 3 lines are a minimal working program, so let's try it.

To execute an entire python program, **always open a new python console** (in spyder, go to console, then click on `open a new python console`, then press F5, - or the green arrow).

Do it.

...

Tadaaaa, *voilà*, you've created your first neuropsydia-based program :)

2. Create a Stroop Task

Contact

For remarks, complaints, suggestions or anything else, please create an issue.

Denomination

The first phase of the Stroop task consists of a training/baseline part referred to as the denomination. Neutral stimuli ("XXXX") are colored in red, blue or green, and the participant must respond as quick as possible.

Let's take our program skeleton:

```
import neuropsydia as n

n.start()
# Here will go the program code
n.close()
```

To display stimuli, we will use the `write()` function that we will insert in a `for` loop.

```
import neuropsydia as n  # Load neuropsydia

n.start()  # Start neuropsydia

n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms

    n.newpage("grey")  # Neutral grey background
    n.write("XXXX")  # Load the stimulus
    n.refresh()  # Display it
    n.response()  # Wait for response

n.close()  # Close neuropsydia
```

Run the program (F5). As you can see, it works, but it's far from being usable. We have to add colors to the stimuli and record the response. Also, add instructions.

```
import neuropsydia as n  # Load neuropsydia
import numpy as np  # Load numpy


n.start()  # Start neuropsydia

n.instructions("Press LEFT when RED, DOWN when GREEN and RIGHT when BLUE.")

n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms
```

```
    stim_color = np.random.choice(["raw_red", "raw_green", "raw_blue"])  # Choose a␣
↪color
    n.newpage("grey")  # Neutral grey background
    n.write("XXXX", style="bold", color=stim_color, size=3)  # Load the stimulus
    n.refresh()  # Display it
    answer, RT = n.response()  # Record response and response time

n.close()  # Close neuropsydia
```

Much better. Now, we're gonna analyze the response (if correct or not) and store them within a dict.

```
import neuropsydia as n  # Load neuropsydia
import numpy as np  # Load numpy

n.start()  # Start neuropsydia

# Initialize data storage
data = {"Stimulus": [],
        "Stimulus_Color": [],
        "Answer": [],
        "RT": [],
        "Condition": [],
        "Correct": []}

#=============================================================================
# Part 1: Denomination
#=============================================================================

n.instructions("Press LEFT when RED, DOWN when GREEN and RIGHT when BLUE.")

n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms

    stim_color = np.random.choice(["raw_red", "raw_green", "raw_blue"])  # Choose a␣
↪color
    stim = "XXXX"
    n.newpage("grey")  # Neutral grey background
    n.write(stim, style="bold", color=stim_color, size=3)  # Load the stimulus
    n.refresh()  # Display it
    answer, RT = n.response()  # Record response and response time

    # Append trial info to
    data["Stimulus"].append(stim)
    data["Stimulus_Color"].append(stim_color)
    data["Answer"].append(answer)
    data["RT"].append(RT)
    data["Condition"].append("Neutral")

    # Categorize the response
    if answer == "LEFT" and stim_color == "raw_red":
        data["Correct"].append(1)
    elif answer == "DOWN" and stim_color == "raw_green":
        data["Correct"].append(1)
```

```
    elif answer == "RIGHT" and stim_color == "raw_blue":
        data["Correct"].append(1)
    else:
        data["Correct"].append(0)

n.close()  # Close neuropsydia

print(data)
```

Conflict

The only thing that will change in the second part is that the stimulus will not always be XXXX, but a color name.

```
import neuropsydia as n  # Load neuropsydia
import numpy as np  # Load numpy

n.start()  # Start neuropsydia

# Initialize data storage
data = {"Stimulus": [],
        "Stimulus_Color": [],
        "Answer": [],
        "RT": [],
        "Condition": [],
        "Correct": []}

#==============================================================================
# Part 1: Denomination
#==============================================================================

n.instructions("Press LEFT when RED, DOWN when GREEN and RIGHT when BLUE.")

n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms

    stim_color = np.random.choice(["raw_red", "raw_green", "raw_blue"])  # Choose a␣
↪color
    stim = "XXXX"
    n.newpage("grey")  # Neutral grey background
    n.write(stim, style="bold", color=stim_color, size=3)  # Load the stimulus
    n.refresh()  # Display it
    answer, RT = n.response()  # Record response and response time

    # Append trial info to
    data["Stimulus"].append(stim)
    data["Stimulus_Color"].append(stim_color)
    data["Answer"].append(answer)
    data["RT"].append(RT)
    data["Condition"].append("Neutral")

    # Categorize the response
    if answer == "LEFT" and stim_color == "raw_red":
```

```
        data["Correct"].append(1)
    elif answer == "DOWN" and stim_color == "raw_green":
        data["Correct"].append(1)
    elif answer == "RIGHT" and stim_color == "raw_blue":
        data["Correct"].append(1)
    else:
        data["Correct"].append(0)

#=============================================================================
# Part 2: Conflict
#=============================================================================

n.instructions("Press LEFT when RED, DOWN when GREEN and RIGHT when BLUE.")

n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms

    stim_color = np.random.choice(["raw_red", "raw_green", "raw_blue"])  # Choose a␣
↪color
    stim = np.random.choice(["RED", "GREEN", "BLUE"])
    n.newpage("grey")  # Neutral grey background
    n.write(stim, style="bold", color=stim_color, size=3)  # Load the stimulus
    n.refresh()  # Display it
    answer, RT = n.response()  # Record response and response time

    # Append trial info to
    data["Stimulus"].append(stim)
    data["Stimulus_Color"].append(stim_color)
    data["Answer"].append(answer)
    data["RT"].append(RT)

    # Categorize the condition
    if stim == "RED" and stim_color == "raw_red":
        data["Condition"].append("Congruent")
    elif stim == "GREEN" and stim_color == "raw_green":
        data["Condition"].append("Congruent")
    elif stim == "BLUE" and stim_color == "raw_blue":
        data["Condition"].append("Congruent")
    else:
        data["Condition"].append("Incongruent")

    # Categorize the response
    if answer == "LEFT" and stim_color == "raw_red":
        data["Correct"].append(1)
    elif answer == "DOWN" and stim_color == "raw_green":
        data["Correct"].append(1)
    elif answer == "RIGHT" and stim_color == "raw_blue":
        data["Correct"].append(1)
    else:
        data["Correct"].append(0)

n.close()  # Close neuropsydia
```

Finally, just before the end (before the close), we can transform the data dict into a `pandas.DataFrame`, that can

then be easily saved. Don't forget to import pandas at the beginning.

```
import neuropsydia as n  # Load neuropsydia
import numpy as np  # Load numpy
import pandas as pd  # Load pandas

n.start()  # Start neuropsydia

# Initialize data storage
data = {"Stimulus": [],
        "Stimulus_Color": [],
        "Answer": [],
        "RT": [],
        "Condition": [],
        "Correct": []}

#=============================================================================
# Part 1: Denomination
#=============================================================================

n.instructions("Press LEFT when RED, DOWN when GREEN and RIGHT when BLUE.")

n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms

    stim_color = np.random.choice(["raw_red", "raw_green", "raw_blue"])  # Choose a␣
↪color
    stim = "XXXX"
    n.newpage("grey")  # Neutral grey background
    n.write(stim, style="bold", color=stim_color, size=3)  # Load the stimulus
    n.refresh()  # Display it
    answer, RT = n.response()  # Record response and response time

    # Append trial info to
    data["Stimulus"].append(stim)
    data["Stimulus_Color"].append(stim_color)
    data["Answer"].append(answer)
    data["RT"].append(RT)
    data["Condition"].append("Neutral")

    # Categorize the response
    if answer == "LEFT" and stim_color == "raw_red":
        data["Correct"].append(1)
    elif answer == "DOWN" and stim_color == "raw_green":
        data["Correct"].append(1)
    elif answer == "RIGHT" and stim_color == "raw_blue":
        data["Correct"].append(1)
    else:
        data["Correct"].append(0)

#=============================================================================
# Part 2: Conflict
#=============================================================================

n.instructions("Press LEFT when RED, DOWN when GREEN and RIGHT when BLUE.")
```

```
n_trials = 10
for trial in range(n_trials):
    n.newpage("grey")  # Neutral grey background
    n.write("+")  # Fixation cross
    n.refresh()  # Display it
    n.time.wait(250)  # Wait 250 ms

    stim_color = np.random.choice(["raw_red", "raw_green", "raw_blue"])  # Choose a␣
↪color
    stim = np.random.choice(["RED", "GREEN", "BLUE"])
    n.newpage("grey")  # Neutral grey background
    n.write(stim, style="bold", color=stim_color, size=3)  # Load the stimulus
    n.refresh()  # Display it
    answer, RT = n.response()  # Record response and response time

    # Append trial info to
    data["Stimulus"].append(stim)
    data["Stimulus_Color"].append(stim_color)
    data["Answer"].append(answer)
    data["RT"].append(RT)

    # Categorize the condition
    if stim == "RED" and stim_color == "raw_red":
        data["Condition"].append("Congruent")
    elif stim == "GREEN" and stim_color == "raw_green":
        data["Condition"].append("Congruent")
    elif stim == "BLUE" and stim_color == "raw_blue":
        data["Condition"].append("Congruent")
    else:
        data["Condition"].append("Incongruent")

    # Categorize the response
    if answer == "LEFT" and stim_color == "raw_red":
        data["Correct"].append(1)
    elif answer == "DOWN" and stim_color == "raw_green":
        data["Correct"].append(1)
    elif answer == "RIGHT" and stim_color == "raw_blue":
        data["Correct"].append(1)
    else:
        data["Correct"].append(0)

df = pd.DataFrame.from_dict(data)  # Convert dict to a dataframe
df.to_csv("data.csv")  # Save data

n.close()  # Close neuropsydia
```

3. Computerize the Beck Depression Inventory (BDI)

Contact

For remarks, complaints, suggestions or anything else, please create an issue.

Code

Now that you're familiar with the basics, we will try the inverse method. Here's the full code of the BDI, that we will try to understand, step by step.

```
import neuropsydia as n  # Load neuropsydia
import pandas as pd


items = {
        1: {0: "I do not feel sad.",
            1: "I feel sad.",
            2: "I am sad all the time and I can't snap out of it.",
            3: "I am so sad and unhappy that I can't stand it."
            },
        2: {0: "I am not particularly discouraged about the future.",
            1: "I feel discouraged about the future.",
            2: "I feel I have nothing to look forward to.",
            3: "I feel the future is hopeless and that things cannot improve."
            },
        3: {0: "I do not feel like a failure.",
            1: "I feel I have failed more than the average person.",
            2: "As I look back on my life, all I can see is a lot of failures.",
            3: "I feel I am a complete failure as a person."
            }
        # I cannot legally show the rest of the questions
        }



n.start()  # Initialize neuropsydia

participant_id = n.ask("Participant ID:", order=1)  # Get participant id
participant_gender = n.ask("Gender:", order=2)  # Get participant's gender
participant_age = n.ask("Age:", order=3)  # get participant's age

n.instructions("Please tell if you agree with each following proposition.")  #␣
↪Instructions


data = {}  # Initialize empty data dict
for item in items:

    data[item] = {}  # For each item, initialize empty sub-dict

    for proposition_number in items[item]:

        question = items[item][proposition_number]  # Current proposition

        n.newpage()
        n.write("\n\n\n" + question, long_text=True)  # Display current proposition
        answer = n.choice([0, 1],
                          overwrite_choices_display=["No", "Yes"],
                          boxes_edge_size=0,
                          boxes_background="teal")

        # Loop control
        if proposition_number == 0:
```

```
            if answer == 1:
                break
        else:
            if answer == 0:
                break

    data[item]["Proposition"] = question  # Store the current proposition
    data[item]["Score"] = proposition_number  # Store the score

# Convert to dataframe and store info
df = pd.DataFrame.from_dict(data, orient="index")
df["Participant_ID"] = participant_id
df["Participant_Gender"] = participant_gender
df["Participant_Age"] = participant_age

# Analysis
df["Score_Total"] = df["Score"].sum()
df.to_csv("BDI_" + participant_id + ".csv")

n.close()  # Close neuropsydia
```

Problem

The thing is that, unlike many other questionnaires, the BDI has a complex scoring procedure, that prevents us from using the `questionnaire()` function.

CHAPTER 2

Examples

Contents:

Questionnaires

State-Trait Anxiety Inventory (STAI-Y)

```
import neuropsydia as n  # Load neuropsydia


questions_dictionary = {

    "Item": {
        1: "I feel calm.",
        2: "I feel secure.",
        3: "I am tense.",
        4: "I feel strained.",
        5: "We're not allowed to reveal all the questions :("  # As it is the last␣
↪dict item, no comma after that.
    },
    "Reverse": {
        1: True,
        2: True,
        3: False,
        4: False,
        5: False
    }
}

n.start()  # Initialize neuropsydia

participant_id = n.ask("Participant ID:", order=1)  # Get participant id
participant_gender = n.ask("Gender:", order=2)  # Get participant's gender
```

```
participant_age = n.ask("Age:", order=3)  # get participant's age

df = n.questionnaire(questions_dictionary,  # The questions
                participant_id=participant_id,
                analog=False,  # Lickert-like
                edges=[1, 4],  # Values underneath
                labels=["Almost never", "Sometimes", "Often", "Almost always"],
                style="blue",  # The cursor's color
                instructions_text="A number of statements which people have used to␣
↪describe themselves are given below. \nSelect the number that indicate how you feel␣
↪right now, that is, at this moment. \nThere are no right or wrong answers. Do not␣
↪spend too much time on any one statement but give the answer which seems to␣
↪describe your present feelings best.")  # Add instructions at the beginning


# Scoring
score = df["Answer"].sum()

# Cutoff based on Crawford et al. (2011). This just for illustration purposes, adapt␣
↪it following your activity.
if score > 56:
    interpretation = "Possible Anxiety"
else:
    interpretation = "No anxiety"

# Add info and save
df["Participant_ID"] = participant_id
df["Participant_Gender"] = participant_gender
df["Participant_Age"] = participant_age
df["Score"] = score
df["Interpretation"] = interpretation
df.to_csv("STAI_" + participant_id + ".csv")

n.close()  # Close neuropsydia
```

Hint: Try to run this questionnaire!

- Can you add all missing questions from a legal version?
- Can you computerize the trait version?

Tasks

Digit Span

```
import neuropsydia as n  # Load neuropsydia
import numpy as np  # For generation of random sequence


n.start()  # Initialize neuropsydia
n.instructions("Listen to the experimenter.")  # Display instructions

# Initialize values
```

```
number_of_fails = 0  # Initial number of errors
span = 2  # Initial span

while number_of_fails < 3:  # Do it while the number of errors is smaller than 3
    sequence = np.random.randint(10, size=span)  # Generate sequence of size span␣
↪with ints ranging from 0 to 9
    good_answer = ""  # Initiate an empty good_answer

    for digit in sequence:  # For every digit in the sequence...
        good_answer = good_answer + str(digit)  # Add the current stimulus to the␣
↪good answer
        n.newpage("grey")  # Load a grey background
        n.time.wait(250)  # Display an empty screen for 250 ms
        n.newpage("grey")  # Load a grey background
        n.write(digit, size=3)  # Load the stimulus
        n.refresh()  # Display the stimulus on screen
        n.time.wait(1000)  # Wait 1000 ms

    # Get answer
    n.newpage("white")
    answer = n.ask("Answer:")

    # Manage result
    if answer == good_answer:
        span = span + 1  # Increase span
        number_of_fails = 0  # Reset value
    else:
        number_of_fails = number_of_fails + 1

n.newpage()  # Load a white background
n.write("Max span: " + str(span-1))  # Write task result
n.refresh()  # Render it on screen
n.time.wait(3000)  # Wait for 3s

n.close()  # Close neuropsydia
```

Hint: Try to run this task!

- Can you change the sequence generation so it contains letters rather than digits?
- Can you change the rules so the sequence length increases every two good answers?
- Can you store the results and save them?

Go/No Go

```
import neuropsydia as n  # Load neuropsydia
import random  # Import the random module
import pandas as pd  # To manipulate and save data
import numpy as np  # To do some maths

n.start()  # Start neuropsydia
n.instructions("Goal: Hit SPACE whenever a GREEN circle appears. \nIf RED, don't␣
↪press anything!")  # Display instructions and break line with \n
n.newpage("grey")  # Fill the screen
```

```
n.countdown()  # Display countdown

# Initialize the data storage with a dictionary containing empty lists
data = {"Trial": [],
        "Stimulus": [],
        "ISI":[],
        "RT":[],
        "Response":[]}

n_trials = 10  # Number of trials
for trial in range(n_trials):  # Iterate over the number of trials
    stimulus = random.choice(["green", "green", "green", "red"])  # Select a stimulus␣
↪type
    ISI = random.randrange(start=250, stop=1250, step=250)  # Select the inter-
↪stimuli interval (ISI)

    n.newpage("grey")  # Fill the screen
    n.write("+")  # Fixation cross
    n.refresh()  # Diplay it on screen
    n.time.wait(ISI)  # Wait

    n.circle(size=2, fill_color=stimulus)  # Display the stimulus (filled with the␣
↪color selected above)
    n.refresh()  # Diplay it on screen
    response, RT = n.response(time_max=1000)  # Wait until 1 s and collect the␣
↪response and its time

    # Categorize the response
    if response == "SPACE" and stimulus == "green":
        response_type = "HIT"  # Hit
    if response != "SPACE" and stimulus == "green":
        response_type = "MISS"  # Miss
    if response == "SPACE" and stimulus == "red":
        response_type = "FA"  # False Alarm
    if response != "SPACE" and stimulus == "red":
        response_type = "CR"  # Correct Rejection

    # Store data by appending each item to its list
    data["Trial"].append(trial)
    data["Stimulus"].append(stimulus)
    data["ISI"].append(ISI)
    data["RT"].append(RT)
    data["Response"].append(response_type)

# Data saving
df = pd.DataFrame.from_dict(data)  # Transform the data dictionary into a proper and␣
↪savable dataframe
df.to_csv("data.csv")  # Save it

# Quick analysis
RTs = df[df['Response']=="HIT"]["RT"]  # Select the Hits' RTs
print("Mean RT: " + str(round(RTs.mean(), 2)))   # Print the mean
print("SD RT: " + str(round(RTs.std(), 2)))  # Print the standard deviation
print("Number of False Alarms: " + str(len(df[df['Response']=="FA"])))  # Print the␣
↪number of intrusions (false alarms)

n.close()  # Close neuropsydia
```

Hint: Try to run this task!

- Can you change the number of trials?
- Can you change the ratio of no go trials?

Flanker

```
import neuropsydia as n  # Load neuropsydia
import pandas as pd  # To manipulate and save data
import numpy as np  # To do some maths

n.start()  # Start neuropsydia
n.instructions("Hit RIGHT or LEFT arrow according to the direction of the CENTRAL␣
↪arrow.")  # Display instructions

# Initialize cache
cache = {}
for possible_angle in [0, 90, 180]:
    cache = n.preload("arrow-left.png", size=2, rotate=possible_angle, cache=cache)
↪# Preload images

# Initialize the data storage with a dictionary containing empty lists
data = {"Trial": [],
        "Trial_Type": [],
        "Stimulus_Orientation": [],
        "RT":[],
        "Response":[]}


n.newpage("grey")  # Fill the screen
n.countdown()  # Display countdown

n_trials = 10  # Number of trials
for trial in range(n_trials):  # Iterate over the number of

    n.newpage("grey")  # Fill the screen
    n.write("+")  # Fixation cross
    n.refresh()  # Diplay it on screen
    n.time.wait(500)  # Wait

    # Trial characteristics
    stimulus_angle = np.random.choice([0, 180])  # select target orientation
    trial_type = np.random.choice(["Congruent", "Neutral", "Incongruent"])  # select␣
↪trial type
    if trial_type == "Congruent":
        distractors_angle = stimulus_angle
    if trial_type == "Incongruent":
        if stimulus_angle == 0:
            distractors_angle = 180
        else:
            distractors_angle = 0
    if trial_type == "Neutral":
        distractors_angle = 90
```

```
    n.image("arrow-left.png", x=-5, size=2, cache=cache, rotate=distractors_angle)  #␣
↪Distractor
    n.image("arrow-left.png", x=-2.5, size=2, cache=cache, rotate=distractors_angle)
↪# Distractor
    n.image("arrow-left.png", x=0, size=2, cache=cache, rotate=stimulus_angle)  #␣
↪Target
    n.image("arrow-left.png", x=2.5, size=2, cache=cache, rotate=distractors_angle)
↪# Distractor
    n.image("arrow-left.png", x=5, size=2, cache=cache, rotate=distractors_angle)  #␣
↪Distractor
    n.refresh()
    response, RT = n.response(time_max=1000)  # Wait until 1 s and collect the␣
↪response and its time


    # Response check
    if (response == "LEFT" and stimulus_angle == 0) or (response == "RIGHT" and␣
↪stimulus_angle == 180):
        response = 1
    else:
        response = 0

    # Store data by appending each item to its list
    data["Trial"].append(trial)
    data["Trial_Type"].append(trial_type)
    data["Stimulus_Orientation"].append(stimulus_angle)
    data["RT"].append(RT)
    data["Response"].append(response)

# Data saving
df = pd.DataFrame.from_dict(data)  # Transform the data dictionary into a proper and␣
↪savable dataframe
df.to_csv("data.csv")  # Save it

# Quick analysis
mean_cong = df[(df["Trial_Type"]=="Congruent") & (df["Response"]==1)]["RT"].mean()
mean_neu = df[(df["Trial_Type"]=="Neutral") & (df["Response"]==1)]["RT"].mean()
mean_incong = df[(df["Trial_Type"]=="Incongruent") & (df["Response"]==1)]["RT"].mean()
print("Mean RT Congruent: " + str(round(mean_cong, 2)))  # Print the mean of congruent
print("Mean RT Neutral: " + str(round(mean_neu, 2)))  # Print the mean of neutral
print("Mean RT Incongruent: " + str(round(mean_incong, 2)))  # Print the mean of␣
↪incongruent

n.close()  # Close neuropsydia
```

Hint: Try to run this task!

- Can you count the number of errors?
- Can you ask for the participant name at the beginning, and save data using it?

CHAPTER 3

Documentation

This part of the documentation details the complete `neuropsydia` for python API.

Core

newpage

refresh

Display

write

image

line

circle

rectangle

Input

response

ask

scale

choice

Meta

questionnaire

Miscellaneous

scale_styles

CHAPTER 4

Installation

Installation can be easily done using `pip`:

```
pip install https://github.com/neuropsychology/Neuropsydia.py/zipball/master
```

Note: A fully working python distribution (including Neuropsydia) can be downloaded here.

CHAPTER 5

Questions? Help? Movie Recommendations?

Do it on the dedicated chat!

-

CHAPTER 6

Index

- genindex
- modindex
- search