## General Naming Convention

Applies to everything you name!

- **lower case**
- use **underscore**, no space or "-"
- names always: **firstname_lastname**
- date always: **yyyymmdd**
- time always: **hhmm** (24h format)
- Animal ID always: **pyrat ID, no "-"** (e.g. OPI7846)

## Raw Data File Naming

File name **has to contain in this order**, separated by **underscore**:

**1) pyrat ID/ plate ID / "ext" for colaboration files** (if not existent, use a placeholder)

**2) date**

**3) time**

4) for microscopy: slide_slice

5) add. information if necessary

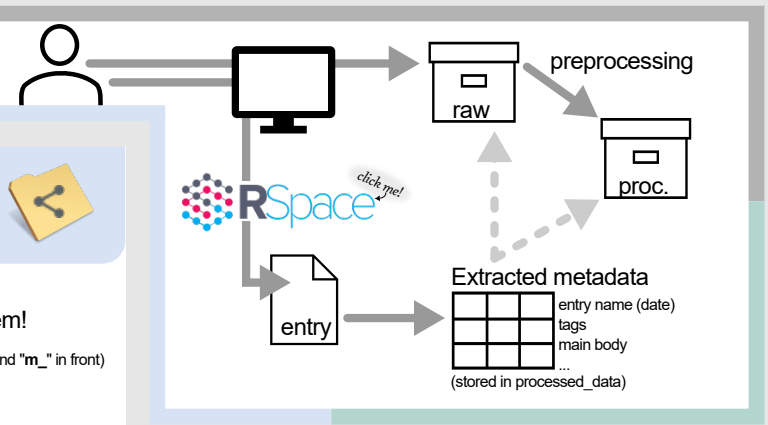
**pyratID_date_time_xxx.ext**
e.g. OPI4590_20260528_0945_ttls.txt

**pyratID_date_time_slide_slice_xxx.ext**
e.g. OPI4590_20260528_0945_5_4_19x_tdTom.txt

## Raw Data Folder Structure

raw_data/ *technique* / *experimenter* / files

- All raw data are stored here
- Read/Write access to own folders; read access to others' folders
- Treat collaboration data as raw data
- Should be cleaned up during offboarding (or before)




## Processed Data Folder Structure

processed_data/ *experimenter* / *project_name* / ...

... **preprocessed** / ...

... **results** / *v00x_analysis* / ...

... **metadata** / ...

... **code** / ... (contains folders: src, dependendies, parameters)

- Preprocessed files are any processed files that further analysis is based on; the folder structure within the preprocessed folder is up to you, but versioning as in the "results" folder may be a good idea in many cases
- Within the results folder, files are stored in versioned folders: v00x_name_of_analysis

## RSpace Lab Group

Experimenter is added by Kristian Reichelt.

### Ontology

- Create all tags first within ontology before using them!
- Contains tags for: - *techniques* (names as in raw_data, append "**m_**" in front)
  - preprocessed, results

### Templates

- Only generic templates that are useful for multiple people
- #experiment_master, mouse_info, ...

### Guidelines

How-Tos for creating tags and templates, creating entries from a template, starting your project folder, exporting metadata

## RSpace Project Group

Experimenter can create it and add others.

### Ontology

- Create all tags first within ontology before using them!
- Contains tags for: - pyrat ID / plate ID (append "**id_**" in front; e.g. OPI5536)
  - TVA identifier

### Protocols (folder)

Contains any kind of protocols used during the experiment.

### Mice (folder)

1 entry per mouse (can be created from mouse_info template): - general information about animal
- entry name: **#pyratID** (e.g. #OPI7755)
- tag the entry using the pyrat ID-tag

### Notebook (= collection of basic documents/entries)

1st entry: experiment master (create from #experiment_master template)

1 entry per session (surgery/ recording/ experiment/ preprocessing/ analysis)

- entry name: **date_time_xxx** (e.g. 20260530_1425_injection)
- tag the entry using the pyrat ID-tag and the technique

Regularly **export your metadata** from RSpace!
(store in your project's metadata folder in processed_data)

## Metadata - Project Information (= README)

- Experiment description
- Information about experimenter
- Experiment Protocols (timeline, protocols & parameters, generated files)
- Info about add. file name parts of recorded files
- Tags assigned to notebook
- Abbreviation dictionary

Information is stored in a project's **experiment master document on RSpace**. → Export as **pdf** and store in metadata folder.

## Metadata - Mouse Information (= Look-up table)

should contain: pyrat ID (e.g. BEC5555), date of birth, sex, genotype, perfusion date, TVA, …

Information is stored in the **mouse_info documents on RSpace**. → Export as **csv**, reformat and store in metadata folder.

## Metadata - Data Pointer (= Look-up table)

- table to find files associated with a specific recording/ analysis session
- should contain: pyrat ID (e.g. BEC5555), date, time (of recorded file)

Information is extracted from entries in project's **notebook on RSpace**. → Export as **csv**, reformat and store in metadata folder.

## Metadata - Codebook

- information about how to understand and use raw and preprocessed files
- should contain:
  - Explanation of what is stored where in file
  - Units of measurement
  - Legend for numeric codes (eg for binary coding)
  - Missing data identifier
  - Abbreviation dictionary
  - How do multiple files relate to each other? (if relevant)
  - How were preprocessed data generated? (if relevant)
  - Version of software data were generated with

Store as pdf/ json/ txt/... in metadata folder.

## Code Writing Guidelines

(not really optional)

### General

- Write your code as modular as possible and be consistent!
- Version tracking with git is mandatory
- Commit each cohesive set of changes that you do separately and write informative commit messages

## Code base structure

processed_data/... /**code** /...

... **src** / ...
contains all utility functions.

... **dependencies** / ...
contains dependencies (e.g. cloned repos).

... **parameters** / ...
contains config/ parameter files.

...

### Naming

- Name variables in a descriptive manner and be consistent.
- Name your scripts in a descriptive manner.
- Name parameters using upper case only (e.g. INDEX = 5).

### Parameter Definition (= variables that are pre-defined)

- Either at top of your main script or in a separate "config" file. (e.g. config.py)
- File paths are parameters, treat them as such.
  Define paths in a relational manner. (define parent directory, derive from it)
  You can code a user interface for path selection.

### Annotations

- Each function needs a header with: input and output definition and purpose of function.
- Add comments to blocks of code to explain what happens.
- If any variables are not self-explanatory, explain them.

### Optional Suggestions/ Ideas

- Take notes in RSpace (or in a local file) for each type of analysis that relate the code used for it to the version index of the corresponding results folder. (e.g. v005_place_coding).
- Consider ways to code more efficiently and check how much time individual parts of your code take to run (especially recommended for modelling and computationally intense analysis).
- In python: regularly check style issues using appropriate libraries (e.g. ruff).

## BeckGroup Github

Lab Code Sharing

click me!

Account contains 4 types of repositories.
Account manager: Kristian Reichelt, Bela Erlinghagen, Lena Gschossmann

### Published code repos

Separate repository per publication. Contains code basis for corresponding publication.

### Analysis code repo

Contains shared functions for analysis and modelling.

### Data management (DM) code repo(s)

Contains code used for data management tasks, such as IEECRSpace repo.

click me!

### Experimental setup code repos

Separate repository per setup. Contains code to run experiments on a setup. Experimenter becomes (partial) admin of IEECR account.

## Publication Code Standards

- Explicitly define language and library versions. (e.g. in a .yml file)
- Each script needs a header with information about people who contributed to it and the final version date.
- Functions necessary for figure-reproduction are summarized in a main script (if possible).
- Codebooks to understand raw or processed data files.
- README file with information regarding:
  - What is the project about?
  - What is the repositories structure?
  - Step-by-step explanation for reproducing figure panels
  - Link to files on BonnData*

**!** Before publication, a co-author checks if he can reproduce figure panels using the code from the repository.

*note: store (raw &) processed files necessary for figure reproduction on BonnData.

## Analysis/ DM Code Standards

- To decide which functions are shared, people present them quickly in lab meetings and check if others could use them.
- Function header should add. contain the author's name and the approx. date.
- Folder structure of repo: **utilities** / **matlab** / files
  utilities / **python** / files
  invivo_imaging / matlab / files
  invivo_imaging / python / files
  behavior / matlab / files
  ...

## Exp. Setup Code Standards

- Explicitly define language and library versions. (e.g. in a .yml file)
- README file.
- Parts list (as far as possible).
- Codebooks (see above) for the raw data that are created with the code.
- Step-by-step guides explaining how to run specific experiments at this setup.
- A Github release should be created before start of experiment (name it with date and version: **yyyymmdd_v00x**)