Cookiecutter: Project Templates and Much More

https://github.com/reka/cookiecutter-tutorial-scipy2024

Everything Set Up?

Check the installed version:

```
cookiecutter --version
```

=> 2.6.0

Try it out:

```
cookiecutter https://github.com/audreyfeldroy/cookiecutter-pypackage.git
```

What's Cookiecutter?

```
cookiecutter https://github.com/audreyfeldroy/cookiecutter-pypackage.git
```

Cookiecutter, the Tool

- a command line tool
- a Python library

https://github.com/cookiecutter/cookiecutter
https://pypi.org/project/cookiecutter/
https://cookiecutter.readthedocs.io/

> Created and led by Audrey Roy Greenfeld, supported by a dedicated team of maintainers and contributors.

https://cookiecutter.readthedocs.io/en/stable/README.html

Cookiecutter Templates

Cookiecutter works with **cookiecutter templates**.

These templates can be:

- local (your file system)
- remote (GitHub)

Cookiecutter and Python

- Written in Python.
- Can be used as a Python library.

BUT

- You can create a project from a template without any coding.
- You can create your own templates without any coding.

This Workshop's Agenda

- Creating a new `cookiecutter` template step by step.
- Conditionals: content depending on variable values.
- Boolean variables.
- Choice variables.
- `pre_prompt` hooks for verifying the environment.
- `pre_gen_project` hooks for user input validation.
- `post_gen_project` hooks for initializing the project environment.
- Creating a template for a Jupyter notebook.

Automate some of your workflows.
Or at least the start of it.

Where Can Cookiecutter Generate a Good Starting Point?

For any directory with text files.

- software projects
- components or modules in a software project
- test data: both for input and expected output
- content: blog posts, articles, etc.
- workshop material
- documentation
- "administration"
- project proposals
- job applications

How to Create a Cookiecutter Template From an Example Directory?

The First Steps

1. Create a new directory for the template.
2. Copy the example directory into the template directory.
3. Create a `cookiecutter.json` file in the template root.
4. Define a variable for the root directory name in `cookiecutter.json`.
5. Rename the root directory.

=> You have a valid cookiecutter template.

Create the directory for the template:

```
mkdir ~/cookiecutter-blog-post
cp -r enum-with-alias ~/cookiecutter-blog-post/
cd ~/cookiecutter-blog-post/
touch ~/cookiecutter-blog-post/cookiecutter.json
```

In `cookiecutter.json`:

```
{
    "slug": "awesome-article"
}
```

5. Rename the root directory: `enum-with-alias` => `{{cookiecutter.slug}}`

```
cd ~/cookiecutter-blog-post/
mv enum-with-alias {{cookiecutter.slug}}
```

How to Create a Cookiecutter Template From an Example Directory?

Continuation

+1. Rename `enum-with-alias.md` as well.

```
cd ~/cookiecutter-blog-post/{{cookiecutter.slug}}
mv enum-with-alias.md {{cookiecutter.slug}}.md
```

Let's set up a git repo:

```
cd ~/cookiecutter-blog-post/
git init
git add .
git commit -m "initial template with slug"
```

Let's Try it Out

Generate a Directory from Our New Template

```
mkdir ~/testing-cookiecutter-blog-post
cd ~/testing-cookiecutter-blog-post
ln -s ~/cookiecutter-blog-post
cookiecutter ~/cookiecutter-blog-post
```

=>

A new directory `awesome-article` has been generated.

Look at the Generated Directory

```
tree awesome-article
```

=>

```
awesome-article/
├── awesome-article.md
├── examples_in_blog_post.py
├── explore_enum_with_alias.py
├── notes.md
└── outline.md

1 directory, 5 files
```

Tools & Tips

- Create a testing directory.
- Create a short symbolic link for the cookiecutter template in the testing directory.
- Set up an alias for `cookiecutter`.
- Use `tree` for an overview of the directory structure.
- VSCode: Use a Jinja extension and set up the file associations in the Workspace Settings.

VSCode Setup

Use a Jinja Extension

Better Jinja by Samuel Colvin.

Configure the File Associations in the Workspace Settings

In `.vscode/settings.json`:

```
{
    "files.associations": {
        "*.md": "jinja-md",
        "*.py": "jinja-py"
    }
}
```

How to Create a Cookiecutter Template From an Example Directory?

Let's Add Some Content

For each piece of content, decide:

- Keep it.
- Remove it.
- Parametrize it.
- Add some conditions.

Some examples:

- **Keep**: The skeleton code in the `.py` files.
- **Remove**: Any enum-specific code in the `.py` files.
- **Parametrize**: The blog post's title.

How to Create a Cookiecutter Template From an Example Directory?

Create a Variable

1. Replace a specific value with a variable.
2. Define the variable in `cookiecutter.json`

Example

1. Replace "Enum with Alias in Python" => `{{cookiecutter.title}}`
2. Define `title` in `cookiecutter.json`:

```
{
    "slug": "awesome-article",
    "title": "Awesome Title"
}
```

Git:

```
cd ~/cookiecutter-blog-post/
git add .
git commit -m "introduce variable title"
```

Testing:

```
cd ~/testing-cookiecutter-blog-post
cookiecutter ~/cookiecutter-blog-post
```

Exercise

Create 2 new variables:

- `reference_docs`
- `howto_docs`

Remove all the content that is specific to the article about enums.

For now, assume that the template is for Python-related blog posts.

What is a Cookiecutter Template?

2 "ingredients":

- a templated project root directory
- a `cookiecutter.json` file

Project Root Directory

- There's always 1.
- Its name must be **templated**. => It must contain min. 1 variable.

The `cookiecutter.json` File

The shortest possible one:

```
{
    "root_directory_name": "example"
}
```

The `cookiecutter.json` File

A longer one:

```
{
  "project_name": "My Awesome Project",
  "project_slug": "{{ cookiecutter.project_name.lower()|replace(' ', '_')|replace('-', '_')|replace('.', '_')|trim() }}",
  "description": "Behold My Awesome Project!",
  "author_name": "Daniel Roy Greenfeld",
  "domain_name": "example.com",
  "email": "{{ cookiecutter.author_name.lower() | trim() |replace(' ', '-') }}@{{ cookiecutter.domain_name.lower() | trim() }}",
  "version": "0.1.0",
  "open_source_license": [
    "MIT",
    "BSD",
    "GPLv3",
    "Apache Software License 2.0",
    "Not open source"
  ],
  "username_type": ["username", "email"],
  "timezone": "UTC",
  "windows": "n",
  "editor": ["None", "PyCharm", "VS Code"],
  "use_docker": "n",
  "postgresql_version": ["16", "15", "14", "13", "12"],
  "cloud_provider": ["AWS", "GCP", "Azure", "None"],
  "mail_service": [
    "Mailgun",
    "Amazon SES",
    "Mailjet",
    "Mandrill",
    "Postmark",
    "Sendgrid",
    "Brevo",
    "SparkPost",
    "Other SMTP"
  ],
  "use_async": "n",
  "use_drf": "n",
```

```
    "frontend_pipeline": ["None", "Django Compressor", "Gulp", "Webpack"],
    "use_celery": "n",
    "use_mailpit": "n",
    "use_sentry": "n",
    "use_whitenoise": "n",
    "use_heroku": "n",
    "ci_tool": ["None", "Travis", "Gitlab", "Github", "Drone"],
    "keep_local_envs_in_vcs": "y",
    "debug": "n"
}
```

cookiecutter-django

2024-07-07 15:02 UTC

Variables

```
    "module_name": "example",
    "function_name": "calculate_{{cookiecutter.module_name}}"
```

Where do we use these variables?

- In the content of the files.
- In file names and directory names.
- In other variables.

Default Value Based on Another Variable

`title` => `slug`

Let's change the `slug`, so that its default value depends on the title.

In `cookiecutter.json`:

```
    "title": "Awesome Title",
    "slug": "{{ cookiecutter.title|lower|replace(' ','-')}}",
```

Testing:

```
cd ~/testing-cookiecutter-blog-post
cookiecutter ~/cookiecutter-blog-post --no-input title="Styling in Pandas"
```

Tools & Tips

- Use the `--no-input` option.
- Use the extra context, like `title="Styling in Pandas"`
- Use realistic test values.

An example:

```
cookiecutter cookiecutter-blog-post \
--no-input \
reference_docs="https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html" \
howto_docs="https://pandas.pydata.org/docs/user_guide/visualization.html" \
title="Plot a Pandas DataFrame"
```

Exercise 2: Default Value Based on Another Variable

Our `cookiecutter-blog-post` template still contains an `explore_enum_with_alias.py` file.

Rename this file, so that its name contains the snake-case version of the `title` instead of `enum_with_alias`.

Exercise 2: A Possible Solution

`cookiecutter.json`:

```
{
    "title": "Awesome Title",
    "slug": "{{ cookiecutter.title|lower|replace(' ','-')}}",
    "module_name": "explore_{{ cookiecutter.slug|replace('-','_')}}",
    "reference_docs": "",
    "howto_docs": ""
}
```

The root directory:

```
tree {{cookiecutter.slug}}
```

=>

```
{{cookiecutter.slug}}
├── {{cookiecutter.module_name}}.py
├── {{cookiecutter.slug}}.md
├── examples_in_blog_post.py
├── notes.md
└── outline.md

1 directory, 5 files
```

Best Practices

- Start small. - No perfectionism.
- Aim for creating a good starting point, not automating every step.
- Use the default values as documentation / examples.

Default Value

```
python_function_name: "my_function"
```

It might serve multiple purposes:

1. Default value.
2. Example value.
3. Some explanation.

How Does It Work?

Cookiecutter uses Jinja

The following are all Jinja templates:

- The content of files.
- File and directory names.
- The values (but not the keys!) of Cookiecutter variables.

How to Use Cookiecutter for Jinja?

With the `_copy_without_render` variable in `cookiecutter.json`.

```
{
    "project_slug": "sample",
    "_copy_without_render": [
        "{{cookiecutter.repo_name}}/templates/*.html",
    ]
}
```

Cookiecutter Docs / Advanced Usage / Copy without Render

2024-07-07

What About Empty Values?

What if we don't have a reference docs or howto docs?

```
cookiecutter cookiecutter-blog-post --no-input reference_docs="" howto_docs="" title="Explore Undocumented Topic"
```

We get an output like this:

> ## More Info
>
> - Reference docs
> - HOWTO docs

Two things to note:

1. Cookiecutter doesn't throw any error because of the missing value.
2. That's probably not the output we want to see.

Conditional Content with Jinja `if` Statements

```
{% if cookiecutter.reference_docs %}* [Reference docs]({{cookiecutter.reference_docs}}){% endif %}
{% if cookiecutter.howto_docs %}
* [HOWTO docs]({{cookiecutter.howto_docs}})
{% endif %}
```

Template Designer Documentation / `if` Statements

Exercise 3: Introduce Similar `if` Statements in `notes.md`

If no Reference docs or HOWTO docs are provided, omit that part of the "## Sources" section.

Exercise 3: Solution

```
## Sources

{% if cookiecutter.reference_docs %}
Reference docs

{{cookiecutter.reference_docs}}
{% endif %}

{% if cookiecutter.howto_docs %}
HOWTO docs

{{cookiecutter.howto_docs}}
{% endif %}
```

Boolean Variables

In `cookiecutter.json`:

```
    "display_optional_section": true,
```

Valid Values

True values:

- 1
- true
- "t"
- "yes"
- "y"
- "on"

False values:

- 0
- false
- "f"
- "no"
- "n"
- "off"

Cookiecutter Docs / Advanced Usage / Boolean Variables

Exercise: Make the Terminology Section Conditional

Introduce a variable to decide whether to display the Terminology section.

Exercise: Solution

In `cookiecutter.json`:

```
    "include_terminology_section": true,
```

In the template files:

```
{% if cookiecutter.include_terminology_section %}
## Terminology

### Glossary
{% endif%}
```

Boolean Variables: The Old Way

Boolean variables were introduced in version 2.2.0.

In older Cookiecutter templates, you might find this syntax:

```
    "git_initial_commit": "y",
```

Boolean Variables: Usages

- Content that isn't always relevant.
- Shorter vs longer way of phrasing.

Advanced Content

- Software project templates: developer tools.

Where Can Cookiecutter Find a Template?

Local:

- local directory
- zip file

Remote:

- GitHub
- GitLab
- BitBucket

Both public and private repositories.

Testing Your Template

pytest-cookies

> pytest-cookies is a pytest plugin that comes with a cookies fixture which is a wrapper for the cookiecutter API for generating projects. It helps you verify that your template is working as expected and takes care of cleaning up after running the tests. 🍪

https://github.com/hackebrot/pytest-cookies
https://pypi.org/project/pytest-cookies/

Testing a Small Template

```python
def test_bake_project_default_single_command(cookies):
    result = cookies.bake()

    assert result.exit_code == 0
    assert result.exception is None
    assert result.project_path.is_dir()


def test_bake_project_multiple_commands(cookies):
    result = cookies.bake(extra_context={"commands": "multiple"})

    assert result.exit_code == 0
    assert result.exception is None
    assert result.project_path.is_dir()
```

Testing a Complex Template

A small sample of the test setup:

```
SUPPORTED_COMBINATIONS = [
    {"username_type": "username"},
    {"username_type": "email"},
    {"open_source_license": "MIT"},
    {"open_source_license": "BSD"},
    {"open_source_license": "GPLv3"},
    {"open_source_license": "Apache Software License 2.0"},
    {"open_source_license": "Not open source"},
    {"windows": "y"},
    {"windows": "n"},
    {"editor": "None"},
    {"editor": "PyCharm"},
    {"editor": "VS Code"},
    {"use_docker": "y"},
    {"use_docker": "n"},
    {"postgresql_version": "16"},
    {"postgresql_version": "15"},
    {"postgresql_version": "14"},
    {"postgresql_version": "13"},
    {"postgresql_version": "12"},
    {"cloud_provider": "AWS", "use_whitenoise": "y"},
    {"cloud_provider": "AWS", "use_whitenoise": "n"},
    {"cloud_provider": "GCP", "use_whitenoise": "y"},
    {"cloud_provider": "GCP", "use_whitenoise": "n"},
    {"cloud_provider": "Azure", "use_whitenoise": "y"},
    {"cloud_provider": "Azure", "use_whitenoise": "n"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Mailgun"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Mailjet"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Mandrill"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Postmark"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Sendgrid"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Brevo"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "SparkPost"},
    {"cloud_provider": "None", "use_whitenoise": "y", "mail_service": "Other SMTP"},
    # Note: cloud_provider=None AND use_whitenoise=n is not supported
```

```
    {"cloud_provider": "AWS", "mail_service": "Mailgun"},
    {"cloud_provider": "AWS", "mail_service": "Amazon SES"},
    {"cloud_provider": "AWS", "mail_service": "Mailjet"},
    {"cloud_provider": "AWS", "mail_service": "Mandrill"},
    {"cloud_provider": "AWS", "mail_service": "Postmark"},
    {"cloud_provider": "AWS", "mail_service": "Sendgrid"},
    {"cloud_provider": "AWS", "mail_service": "Brevo"},
    {"cloud_provider": "AWS", "mail_service": "SparkPost"},
    {"cloud_provider": "AWS", "mail_service": "Other SMTP"},
    {"cloud_provider": "GCP", "mail_service": "Mailgun"},
    {"cloud_provider": "GCP", "mail_service": "Mailjet"},
    {"cloud_provider": "GCP", "mail_service": "Mandrill"},
    {"cloud_provider": "GCP", "mail_service": "Postmark"},
    {"cloud_provider": "GCP", "mail_service": "Sendgrid"},
    {"cloud_provider": "GCP", "mail_service": "Brevo"},
    {"cloud_provider": "GCP", "mail_service": "SparkPost"},
    {"cloud_provider": "GCP", "mail_service": "Other SMTP"},
    {"cloud_provider": "Azure", "mail_service": "Mailgun"},
    {"cloud_provider": "Azure", "mail_service": "Mailjet"},
    {"cloud_provider": "Azure", "mail_service": "Mandrill"},
    {"cloud_provider": "Azure", "mail_service": "Postmark"},
    {"cloud_provider": "Azure", "mail_service": "Sendgrid"},
    {"cloud_provider": "Azure", "mail_service": "Brevo"},
    {"cloud_provider": "Azure", "mail_service": "SparkPost"},
    {"cloud_provider": "Azure", "mail_service": "Other SMTP"},
    # Note: cloud_providers GCP, Azure, and None
    # with mail_service Amazon SES is not supported
    {"use_async": "y"},
    {"use_async": "n"},
    {"use_drf": "y"},
    {"use_drf": "n"},
    {"frontend_pipeline": "None"},
    {"frontend_pipeline": "Django Compressor"},
    {"frontend_pipeline": "Gulp"},
    {"frontend_pipeline": "Webpack"},
    {"use_celery": "y"},
    {"use_celery": "n"},
    {"use_mailpit": "y"},
    {"use_mailpit": "n"},
    {"use_sentry": "y"},
    {"use_sentry": "n"},
    {"use_whitenoise": "y"},
```

```
    {"use_whitenoise": "n"},
    {"use_heroku": "y"},
    {"use_heroku": "n"},
    {"ci_tool": "None"},
    {"ci_tool": "Travis"},
    {"ci_tool": "Gitlab"},
    {"ci_tool": "Github"},
    {"ci_tool": "Drone"},
    {"keep_local_envs_in_vcs": "y"},
    {"keep_local_envs_in_vcs": "n"},
    {"debug": "y"},
    {"debug": "n"},
]

UNSUPPORTED_COMBINATIONS = [
    {"cloud_provider": "None", "use_whitenoise": "n"},
    {"cloud_provider": "GCP", "mail_service": "Amazon SES"},
    {"cloud_provider": "Azure", "mail_service": "Amazon SES"},
    {"cloud_provider": "None", "mail_service": "Amazon SES"},
]
```

Testing a Complex Template

A small sample of the tests:

```
@pytest.mark.parametrize("context_override", SUPPORTED_COMBINATIONS, ids=_fixture_id)
def test_project_generation(cookies, context, context_override):
    """Test that project is generated and fully rendered."""

    result = cookies.bake(extra_context={**context, **context_override})
    assert result.exit_code == 0
    assert result.exception is None
    assert result.project_path.name == context["project_slug"]
    assert result.project_path.is_dir()

    paths = build_files_list(str(result.project_path))
    assert paths
    check_paths(paths)


@pytest.mark.parametrize("context_override", SUPPORTED_COMBINATIONS, ids=_fixture_id)
def test_ruff_check_passes(cookies, context_override):
    """Generated project should pass ruff check."""
    result = cookies.bake(extra_context=context_override)

    try:
        sh.ruff("check", ".", _cwd=str(result.project_path))
    except sh.ErrorReturnCode as e:
        pytest.fail(e.stdout.decode())
```

https://github.com/cookiecutter/cookiecutter-django/blob/master/tests/test_cookiecutter_generation.py

2024-07-07 15:36 UTC

Inject a Timestamp

```
    "timestamp":  "{% now 'utc', '%Y-%m-%dT%H-%M-%S' %}",
```

How Does It Work?

Default Value Using a Jinja Extension

```
    "timestamp":  "{% now 'utc', '%Y-%m-%dT%H-%M-%S' %}",
```

Cookiecutter Docs / Template Extensions

Best Practices

- Add a README
- Consider providing example json files with frequent parameter combinations.
- Follow conventions in variable naming. (This makes the `default_context` more useful.)

Hooks

> Cookiecutter hooks are scripts executed at specific stages during the project generation process.

Cookiecutter Docs / Advanced Usage / Hooks

- When?
- What?
- Using Cookiecutter variables in hooks.

When Can a Hook Run?

Stages

- `pre_prompt`
- `pre_gen_project`
- `post_gen_project`

Where Do the Hooks Run?

- `pre_prompt`: "in the root directory of a copy of the repository directory"
- `pre_gen_project`: in the root directory of the generated project
- `post_gen_project`: in the root directory of the generated project

Programming Languages Supported

- Python
- Shell
- Shell => lot of other stuff

Shell Hooks

Shell hooks need a shebang.

```
ERROR: Stopping generation because post_gen_project hook script didn't exit successfully
Hook script failed, might be an empty file or missing a shebang
```

Exercise: Create a `post_gen_project` Hook to Initialize a Git Repo

In a `post_gen_project` hook:

1. Initialize a GitHub repo.
2. Add an initial commit.

Exercise: Solution

```
tree cookiecutter-blog-post
```

=>

```
cookiecutter-blog-post
├── cookiecutter.json
├── {{cookiecutter.slug}}
│   ├── {{cookiecutter.module_name}}.py
│   ├── {{cookiecutter.slug}}.md
│   ├── examples_in_blog_post.py
│   ├── notes.md
│   └── outline.md
└── hooks
    └── post_gen_project.sh

3 directories, 7 files
```

In `hooks/post_gen_project.sh`:

```
#!/bin/bash

git init
git add .
git commit -am "project skeleton generated with cookiecutter-blog-post"
```

`pre_prompt` Hook Use Cases

- Verify the environment, especially for software project templates.
- Display a message to the user.

`pre_gen_project` Hook Use Cases

- Validate the input.

Validate That the Title Has Min. 3 Characters

We can access the variable values via Jinja syntax in the hooks as well:

```
title = '{{cookiecutter.title}}'
```

Which means, we can write a validation like this:

```
import sys


def validate_params():
    title = '{{cookiecutter.title}}'
    if len(title) < 3:
        print("Title must be minimum 3 characters long.")
        sys.exit(1)


if __name__ == "__main__":
    validate_params()
```

Exercise: Validate That the Module Name is Valid

Some possible validations:

- The module name doesn't contain a `-`.
- Length validation.
- Regex.

`post_gen_project` Hook Use Cases

- Initialize the project that has been created.
- Move around the files to various places.
- Delete stuff that is unnecessary based on the chosen variable values.

Initialize the Project

- `git init`
- Create a virtual environment and install dependencies.
- Run formatters, e.g. `black`.

Move Around Files

`cookiecutter` can be used not only for whole projects.

You can use the `post_gen_project` hook to move the files created to their place.

For example, if you create a new functionality in a big software project:

- Move the module created to the respective package.
- Move the test file to the test package.
- Move the documentation to the docs dir.

Best Practices

The same as for scripts generally.

- Modularity.
- Display messages to the user, especially for long-running operations.
- For more complex hooks, you might prefer Python.
- Error handling.

> Hooks should be robust and handle errors gracefully. If a hook exits with a nonzero status, the project generation halts, and the generated directory is cleaned.

Cookiecutter Docs / Advanced Usage / Hooks / Hook Execution

Where Can the Variable Values Come From?

What we've already seen:

- CLI input
- Default values in the `cookiecutter.json` file.
- `extra_context` used on the command line and by the tests

Further places where the variable value can come from:

- `--replay` option
- `--replay-file` option
- user configuration

`--replay` Option

> On invocation Cookiecutter dumps a json file to ~/.cookiecutter_replay/ which enables you to replay later on.
>
> In other words, it persists your input for a template and fetches it when you run the same template again.

Cookiecutter Docs / Advanced Usage / Replay Project Generation

How to use this persisted input?

```
cd ~/testing-cookiecutter-blog-post
cookiecutter cookiecutter-blog-post --replay
```

`--replay` Option

How does this persisted input look like?

`~/.cookiecutter_replay/cookiecutter-blog-post.json`

```
{
  "cookiecutter": {
    "title": "xx",
    "slug": "xx",
    "module_name": "explore_xx",
    "include_terminology_section": true,
    "reference_docs": "",
    "howto_docs": "",
    "_template": "cookiecutter-blog-post",
    "_output_dir": "/home/reka/testing-cookiecutter-blog-post",
    "_repo_dir": "cookiecutter-blog-post",
    "_checkout": null
  },
  "_cookiecutter": {
    "title": "xx",
    "slug": "{{ cookiecutter.title|lower|replace(' ','-')}}",
    "module_name": "explore_{{ cookiecutter.slug|replace('-','_')}}",
    "include_terminology_section": true,
    "reference_docs": "",
    "howto_docs": ""
  }
}
```

- The same parameters as in `cookiecutter.json` and some private variables wrapped in `cookiecutter`.
- The same parameters as in `cookiecutter.json` wrapped in `_cookiecutter`.

`--replay-file` Option

- Pre-fill some values.
- Define parameter combinations that usually occur together.

```
cd ~/testing-cookiecutter-blog-post
cookiecutter cookiecutter-blog-post --replay-file custom-replay-file.json
```

`--replay-file` Option: Define a Replay File for Regex Blog Posts

In `replay-config/regex.json`:

```
{
    "cookiecutter": {
        "reference_docs": "https://docs.python.org/3/library/re.html",
        "howto_docs": "https://docs.python.org/3/howto/regex.html"
    }
}
```

Generate a blog post directory with this config:

```
cd ~/testing-cookiecutter-blog-post
cookiecutter cookiecutter-blog-post --replay-file cookiecutter-blog-post/replay-configs/regex.json
```

Exercise: Create a Custom Replay File for `datetime`-related Blog Posts

- Create a custom replay file for `datetime`-related blog posts in the `replay-configs` directory.
- Note that the Python Documentation contains only a reference documentation, but no HOWTO.

https://docs.python.org/3/library/datetime.html

Cookiecutter Config

User Config Sources

- `$HOME/.cookiecutterrc`
- `--config-file` CLI option

Cookiecutter Docs / Advanced Usage / User Config

What to put into the Cookiecutter Config?

`default_context`

Variables used by multiple different templates.

Some examples:

- user name
- email address

Other Settings

- Abbreviations for frequently used templates
- `replay_dir`
- `cookiecutters_dir`

How to Create a Template from a Jupyter Notebook?

Very similar to creating a template from a directory.

Some additional steps:

- Remove the cell outputs.
- Initialize a virtual environment in the `post_gen_project` hook.

Thank You

Thanks for your participation.

rekaa518@proton.me