

# Great on their Own, Even Better Together

Application Development with  
Python, Typer, and Poetry

Gregory M. Kapfhammer

CodepaLOUsa 2021

# Okay, what is this about?

## Key Questions

What are the **benefits** and **challenges** associated with using the Python language, Typer, and Poetry for creating command-line applications?

## Intended Audience

An **adventuresome** technology enthusiast who wants to explore how both a new **paradigm** and software **tools** can improve their development skills!



**Let's create a command-line application in Python!**

# Why focus on Python programming?

## Prevalence of Python

Python is consistently ranked as one of the **top programming languages** for web development, data science, machine learning, and general programming

## Command-Line Interface

Programmers who start using Python through Jupyter notebooks may need to create **tools** and **servers** that require a command-line interface



**What is challenging about programming in Python?**



# Creating virtual environments

- virtualenv
- venv
- pipenv



# Publishing packages to PyPI

- twine
- flit
- setup.py



# Making command-line interfaces

- argparse
- fire
- click

# What are the downsides of these tools?



**virtualenv uses the requirements.txt file**

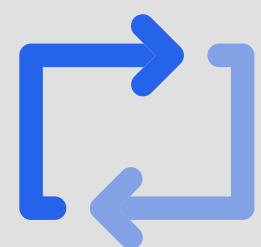


**twine requires use of complicated setup.py file**



**argparse does not verify command-line arguments**

# How to easily create command-line tools using modern Python?



**Typer:**

`https://typer.tiangolo.com/`



**Poetry:** `https://python-`

`poetry.org/`

# Typer

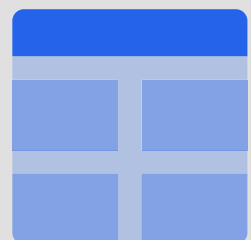
- *Annotations* : assign types to functions accepting arguments
- *Productivity* : types aid in the creation of the interface
- *Checking* : confirm that inputs match expected types

# Poetry

- *Environments* : manage dependencies in isolation
- *Package* : create a stand-alone executable application
- *Publish* : expedite and simplify the release of program to PyPI



**New way to manage application dependencies**



**Adjust to the challenge of adding type annotations**



**Easy command-line interface with Typer**



**Manage, package, and release with Poetry**



**AnalyzeActions/WorkKnow**



# Creating the Application with Poetry

`poetry new workknow`

```
├── coverage.xml
├── poetry.lock
├── pyproject.toml
├── README.md
├── tests
│   ├── __init__.py
│   ├── test_analyze.py
│   ├── test_constants.py
│   └── test_request.py
├── workknow
│   ├── __init__.py
│   ├── analyze.py
│   ├── concatenate.py
│   ├── configure.py
│   ├── constants.py
│   ├── display.py
│   ├── environment.py
│   ├── files.py
│   └── main.py
```

- Create a simple directory structure
- Default support for testing with Pytest
- Store separate modules in directory
- The main file stores command-line interface
- The `pyproject.toml` file stores dependencies
- The `poetry.lock` file pins dependencies

# Application

```
[tool.poetry.dependencies]
python = "^3.8"
typer = {extras = ["all"],
         version = "^0.3.2"}
rich = "^10.5.0"
requests = "^2.25.1"
python-dotenv = "^0.18.0"
pandas = "^1.3.0"
giturlparse = "^0.10.0"
types-pytz = "^2021.1.0"
PyGithub = "^1.55"
pluginbase = "^1.0.1"
tabulate = "^0.8.9"
types-tabulate = "^0.8.1"
pingouin = "^0.3.12"
```

# Development

```
[tool.poetry.dev-dependencies]
pytest = "^5.2"
pylint = "^2.6.0"
black = "^20.8b1"
pydocstyle = "^5.1.1"
flake8 = "^3.8.4"
taskipy = "^1.8.1"
pytest-cov = "^2.11.1"
mypy = "^0.910"
pandas-stubs = "^1.1.0"
types-requests = "^2.25.0"
responses = "^0.13.3"

[tool.poetry.scripts]
workknow = "workknow.main:cli"
```



**Poetry installs packages into the virtual environment**

# Command-Line Interface with Typer

```
import typer
cli = typer.Typer()
@cli.command()
def download(
    repo_urls: List[str],
    repos_csv_file: Path = typer.Option(None),
    results_dir: Path = typer.Option(None),
    env_file: Path = typer.Option(None),
):
```



See [AnalyzeActions/WorkKnow](#) for details!

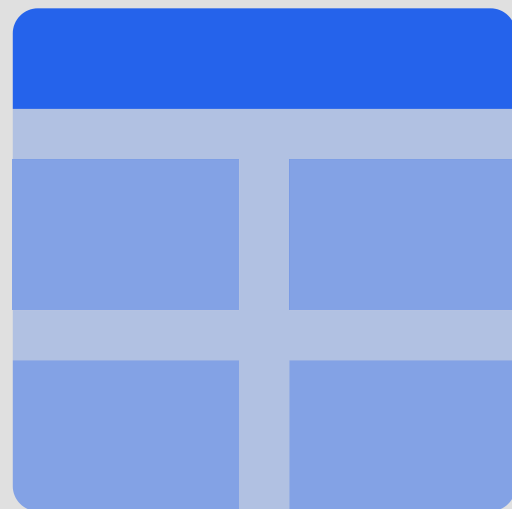
# Adding Extra Commands with Typer

```
import typer
cli = typer.Typer()
@cli.command()
def analyze(
    results_dir: Path = typer.Option(None),
    plugin: str = typer.Option(""),
    save: bool = typer.Option(False),
    debug_level: debug.DebugLevel =
        debug.DebugLevel.ERROR,
):
```

AnalyzeActions/WorkKnow **contains several**

# Command-Line Interface Documentation

```
Usage: workknow download [OPTIONS] REPO_URLS...
       Download the GitHub Action workflow history of repositories.
Arguments:
  REPO_URLS...  [required]
Options:
  --repos-csv-file PATH
  --results-dir PATH
  --env-file PATH
  --peek / --no-peek                [default: False]
  --save / --no-save                [default: False]
  --debug-level [DEBUG|INFO|WARNING|ERROR|CRITICAL]
                                       [default: ERROR]
  --help                            Show this message and exit.
```



- Using type annotations, Typer can:
  - automatically generate all menus
  - perform error checking on all arguments
  - convert all arguments to the correct type

# Running the Program with Poetry

```
poetry run workknow download --repos-csv-file [CSV File]
                              --env-file [ENV File]
                              --results-dir [Results Directory]
                              --debug-level ERROR
                              --save
                              --combine
```



- Poetry takes the following steps:
  - load dependencies into virtual environment
  - locate the "script" variable that defines main
  - invoke the main function and pass control



**What other cool features does Poetry support?**

# Specifying Tasks with Poetry

```
[tool.taskipy.tasks]
black = { cmd = "black workknow tests --check" }
coverage = { cmd = "pytest -s --cov-config .coveragerc [...]" }
flake8 = { cmd = "flake8 workknow tests" }
mypy = { cmd = "poetry run mypy workknow" }
pydocstyle = { cmd = "pydocstyle workknow tests" }
pylint = { cmd = "pylint workknow tests" }
test = { cmd = "pytest -x -s" }
```



- Combining Poetry with Taskipy offers:
  - task specification in pyproject.toml file
  - task execution through use of Poetry
  - "poetry run task all" to run all tasks

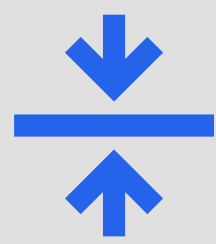


**What are the benefits of running these tasks?**

# Benefits of type checking and code formatting?



**MyPy: Install and run a type checker on code modules**



**Black: Install and run a code formatter for all Python files**



# Defect Detection with Type Checker

```
def create_results_zip_file(
    results_dir: Path, results_files: List[str]
) -> None:
    """Make a .zip file of all results."""
    with zipfile.ZipFile(
        "results/All-WorkKnow-Results.zip",
        "w",
    ) as results_zip_file:
        for results_file in results_files:
            results_zip_file.write(results_file)
```



# Automated Type Checker Feedback

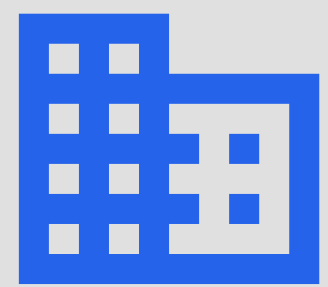
Argument of type "List[str]" cannot be assigned to parameter "filename" of type "StrPath" in function "write"

```
with zipfile.ZipFile(
    "results/All-WorkKnow-Results.zip",
    "w",
) as results_zip_file:
    for results_file in results_files:
        results_zip_file.write(results_files)
```

results\_file



# How to build and publish a Python package?



**Build: create package in standard format**



**Publish: publicly release the package to PyPI**

# Publishing a Package to PyPI

## Poetry Build

Creates the project's "wheel", the standard format for Python packages. User installation of the .whl is possible. Program works without use of Poetry!

## Poetry Publish

After creating a PyPI authorization token and configuring Poetry to use it, the publish command makes it available to everyone through PyPI!



**Program is available for installation with pip or pipx!**

# Challenges

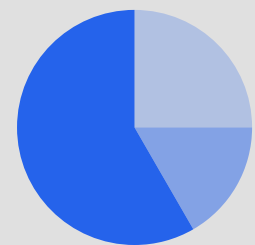
- Not stand-alone binary, so program needs Python to run
- Poetry and Typer are relatively new tools, so defects are possible
- Typer only works if you use type annotations, so extra work needed

# Benefits

- Poetry seamlessly manages dependencies and environments
- Typer automatically creates the command-line interface
- Poetry makes task running and publishing to PyPI effortless

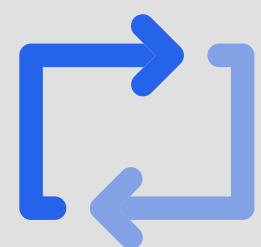


**Two packages to build command-line tools in Python!**



**Quick environments, dependencies, and releases!**

# Best way to easily create command-line tools using modern Python?



**Typer:**

`https://typer.tiangolo.com/`



**Poetry:** `https://python-`

`poetry.org/`

# Great resources for learning more about these Python tools?

```
https://typer.tiangolo.com/tutorial/package/
```

```
https://realpython.com/effective-python-environment/
```



**Share your experiences with the Python community!**

# Tool Development with Python

**Typer and Poetry effectively work together!**



**Programmers define types for functions**



**Create program's command-line with Typer**



**Poetry handles dependencies and releases**



# Tool Development with Python

Typer and Poetry provide an "opinionated" option



AnalyzeActions/WorkKnow



<https://www.gregorykapfhammer.com/>



gkapfham/codepalousa2021-presentation-typer