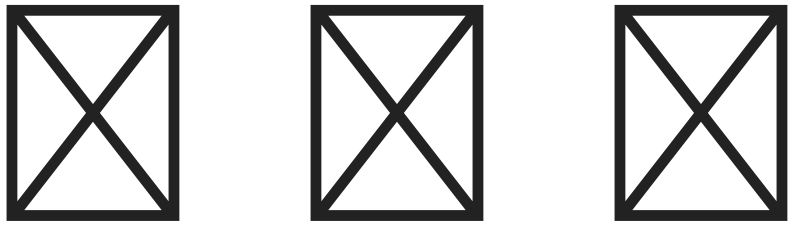
WRITING COMMAND LINE APPLICATIONS THAT CLICK

Instructions: http://bit.ly/pycon-click

dave@forgac.com - @tylerdave

WELCOME TO CLEVELAND!



THE GOAL

Learn to write "well-behaved" command line applications in Python using `click`.

Well behaved?

THE UNIX PHILOSOPHY

- Write programs that do one thing and do it well.
- Write programs that work together.
- Write programs to handle text streams, because that is a universal interface.

DO ONE THING

WORK TOGETHER

HANDLE TEXT STREAMS

Why?

WHY WRITE CLIS?

WHY PYTHON? ☒

WHY `click`?

https://click.palletsprojects.com/why/

WELL BEHAVED CLIS

ARGUMENTS & OPTIONS

```
# Arguments:
exmaple argumentA argumentB

# Options:
example --count 3

# Option flags:
exmaple --verbose

# Arguments and Options:
example --verbose --count3 argumentA argumentB
```

stdin/stdout/stderr

```
# Read stdin:
echo "input text" | example

# Write stdout:
example > outfile.txt

# Write stdout w/ stderr:
example > outfile.txt
INFO: Generating output

# Redirect both:
example > outfile.txt 2> errfile.txt
```

EXIT CODE

```
if ! [ -x "$(example argumentA)" ]; then
  echo 'Error: running example failed.' >&2
  exit 1
fi
```

Ctrl-c / SIGNALS

```
INFO: Running long process...
[Ctrl-c]
INFO: Shutting down gracefully...
```

CONFIGURATION

- Mac OS X:
  `~/Library/Application Support/Foo Bar`
- Unix:
  `~/.config/foo-bar`
- Windows (non-roaming):
  `C:\Users\<user>\AppData\Local\Foo Bar`

COLORS

```
cat good_outfile.txt
Output should look like this.

cat bad_outfile.txt
Output \u001b[31;1mshouln't\u001b[30;1m look like this.
```

CLICK

https://click.palletsprojects.com/

```python
import click

@click.command()
@click.option('--count', default=1,
              help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of
    COUNT times."""
    for x in range(count):
        click.echo('Hello %s!' % name)

if __name__ == '__main__':
    hello()
```

```
Usage: cl.py [OPTIONS]

  Simple program that greets NAME for a total of COUNT times.

Options:
  --count INTEGER  Number of greetings.
  --name TEXT      The person to greet.
  --help           Show this message and exit.
```

THE TUTORIAL

INSTALLATION

- This repo is a Python package
- Installs commands:
  - `pycon` and `tutorial`
  - Lessons: `part01`, `part02`, `part03`
  - `pytest` and `cookiecutter`
- `pycon verify` will test your environment

TUTORIAL-RUNNER

```
Usage: tutorial [OPTIONS] COMMAND [ARGS]...

Click tutorial runner.

Options:
--help Show this message and exit.
Commands:
check    Check your work for the current lesson.
init    (Re-)Initialize the tutorial
lesson    Switch to a specific lesson
peek    Look at the solution file without overwriting
solve    Copy the solution file to the working file.
status    Show the status of your progress.
version    Display the version of this command.
```

INITIALIZE

```
tutorial init
Tutorial initialized! Time to start your first lesson!
```

SHOW LESSON

```
tutorial lesson

Currently working on Part 01, Lesson 01 - Hello, PyCon!

Working file: lessons/part_01/cli.py
       Tests: lessons/part_01/tests/01_hello.py
     Command: part01
Related docs: https://click.palletsprojects.com/en/7.x/quickst

Objectives:
* Learn how to use the `tutorial` command to run and check les
* Make the tests for the first lesson pass by editing the work
```

Try running the command

```
part01
Hello.
```

```
part01 --help
Usage: part01 [OPTIONS]

Options:
  --help  Show this message and exit.
```

CHECK YOUR WORK

- Runs tests
- Outputs assertion results
- Proceeds to next lesson upon success

```
tutorial check
```

NEED A HINT?

Display a solution file that makes test pass:

```
tutorial peek
```

SOLVE

```
tutorial solve
This will make a backup of the working file and then copy the
  Working file: lessons/part_01/cli.py
   Backup file: lessons/part_01/cli.2019-04-29.12-54-03.py
 Solution file: lessons/part_01/solutions/cli_01_hello.py
Do you wish to proceed? [y/N]:
```

Then run check to advance

```
tutorial check
```

OTHER COMMANDS

Check overall status:

```
tutorial status
```

Skip to specific lesson:

```
tutorial lesson --part 1 --lesson 1
```

```
tutorial lesson -p1 -l1
```

TUTORIAL

PART 01:
COMMAND PARSING

HELLO, PYCON!

```
import click

@click.command()
def cli():
    print("Hello.")
```

01-01: HELLO, PYCON!

- Learn how to use the `tutorial` command to run and check lessons.
- Make the tests for the first lesson pass by editing the working file.

ARGUMENTS

```
@click.command()
@click.argument("names", nargs=1)
def cli(name):
    print(f"Hello, {name}.")
```

01-02: ARGUMENTS

- Make the command accept a positional NAME argument
  - accept any number of values
  - print "Hello, NAME!" on a new line for each name given
  - output nothing if no arguments are passed (noop)

OPTIONS

```
@click.command()
@click.argument("name")
@click.option("--count", "-c", default=1)
@click.option("--green", is_flag=True)
@click.option("--debug/--no-debug")
def cli(name):
    ...
```

01-03: OPTIONS

- Add multiple options:
  - Add `--greeting` to specify greeting text
    - With a short alias: `-g`
    - Default to "Hello" if no greeting is passed
  - Add a `--question` option as a flag that doesn't take a value
    - If passed, end the greeting with "?"
    - If not passed, end the greeting with "!" "

HELP DOCUMENTATION

```
@click.command()
@click.option("--count", help="Number of times to print greeti
def cli(count):
    """Print a greeting."""
    ...
```

01-04: HELP DOCUMENTATION

- Document your script
  - Add general command usage help
  - Add help text to the `--greeting` and `--question` options

INPUT VALIDATION

```
@click.command()
@click.option("--example", default=1)
@click.option("--another", type=int)
@click.option("--color", type=click.Choice("red", "green", "bl
def cli(example, another, color):
    ...
```

01-05: INPUT VALIDATION

- Add new options to learn how type validation works
  - Output "int: {VALUE}" if `--int-option`
  - Output "float: {VALUE}" if `--float-option`
  - Output "bool: {VALUE}" if `--bool-option`
  - Add `--choice-option`
    - Validate values are one of "A", "B", "C"
    - Output "choice: {VALUE}" if value passed

PART 02:
INPUT / OUTPUT

ECHO

```
@click.command()
def cli():
    click.echo("I'm about to print...", err=True)

    click.echo("Hello!")

    click.echo(click.style("Green text!", fg="green"))
    # equivalent:
    click.secho("Green text!", fg="green")
```

02-01: ECHO

- Customize output destination and formatting
  - Make "Hello!" print to stdout
  - Make "Printing..." print to stderr
  - Add a `--red` option that makes output text red when passed

FILE I/O

```
@click.command()
@click.argument("infile", type=click.File("r"), default="-")
def cli(infile):
    text = infile.read()
```

02-02: FILE I/O

- Read from and write to files or stdin/stdout depending on arguments
  - Read the input source and write the contents to the output
  - Accept an input file argument, reading from stdin by default (using – arg)
  - Accept an output file argument, writing to stdout by default (using – arg)
  - Find the length of the input data and print a message to stderr

PART 03:

NESTED COMMANDS

COMMAND GROUPS

```
@click.group()
def example_command():
    """I'm an example command."""

@example_command.command()
def example_subcommand():
    """Says hi."""
    click.echo("Hello, world!")
```

03-01: COMMAND GROUPS

- Make a command that has subcommands
  - Add `hello` subcommand that prints "Hello!"
  - See that trying to run nonexistent subcommands results in an error

SHARING CONTEXTS

```
@click.group()
@click.pass_context
def example_command(ctx):
    ctx.obj = {"setting": "value"}

@example_command.command()
@click.pass_context
def example_subcommand(ctx):
    sttings = ctx.obj
    click.echo(settings.get("setting"))

@example_command.command()
@click.pass_obj
def another_subcommand(obj):
    click.echo(obj.get("setting"))
```

03-02: SHARING CONTEXTS

- Learn how parameters are handled by the group and by subcommands
  - Pass `--verbose` group option to `hello` subcommand via `pass_context`
  - Add a new `goodbye` subcommand
  - Pass `--verbose` group option to `goodbye` via `pass_obj`

PART 04:
PROJECTS & PACKAGING

04-01: CREATE A PROJECT

- Use `cookiecutter` to create a new project
- Follow the prompts to enable the CLI

04-01: COOKIECUTTER EXAMPLE

```
full_name [Audrey Roy Greenfeld]: Dave Forgac
email [audreyr@example.com]: tylerdave@tylerdave.com
github_username [audreyr]: tylerdave
project_name [Python Boilerplate]: Example CLI
project_slug [example_cli]:
project_short_description [Python Boilerplate.]: An example CL
pypi_username [tylerdave]: tylerdave
version [0.1.0]: 0.0.1
use_pytest [n]: y
use_pypi_deployment_with_travis [y]: n
Select command_line_interface:
1 - Click
2 - No command-line interface
Choose from 1, 2 (1, 2) [1]: 1
```

04-02: PACKAGE LAYOUT

- Explore package layout
- `setup.py`
- `setup.cfg`
- $PACKAGE_NAME/`cli.py`

04-03: DEVELOPMENT

- Create a virtualenv
- Install the package in editable mode
- See changes reflected

04-04: TESTING

- Update tests to match CLI output

04-05: BUILD & PUBLISH

- Build distribution files
- See how to publish on PyPI

PART 05:
EXTRAS

EXAMPLES

- Pagination
- Progress Bars
- Pagination
- Launch Editor
- Handle `ctrl-c`

THANK YOU!

FEEDBACK / QUESTIONS?

dave@forgac.com

@tylerdave