Using functions for automation

COMMAND LINE AUTOMATION IN PYTHON

Noah Gift

Lecturer, Northwestern & UC Davis & UC Berkeley | Founder, Pragmatic AI Labs

datacamp

Functions are units of work

- functions are units of work

```
def work()
    """Performs work"""
    return 1+1
```

```
In [1]: work()
Out[1]: 2
```

Accepting input into a function

- function that accepts input

```
def more_work(x,y):
    """Adds two variables"""

    return x+y
```

```
In [2]: more_work(5,6)
Out[2]: 1
```

```
more_work(9,6)
Out[3]: 15
```

Decorators are also functions

```
from functools import wraps
import time
def instrument(f):
    @wraps(f)
    def wrap(*args, **kw):
        ts = time.time()
        result = f(*args, **kw)
        te = time.time()
        print(
            f"function: {f.__name__}, args: [{args}, {kw}] took: {te-ts} sec")
        return result
    return wrap
```

How does a decorator work?

```
from functools import wraps

def do_nothing_decorator(f):
    @wraps(f)
    def wrapper(*args, **kwds):
        print('INSIDE DECORATOR: This is called before function')
        return f(*args, **kwds)
    return wrapper

@do_nothing_decorator
def hello_world():
    """This is a hello world function"""

    print("Hello World Function")
```

Using a hello world decorator

```
# Call the decorated function
hello_world()
```

```
INSIDE DECORATOR: This is called before function
Hello World Function
```

```
# Name is preserved
print(f"Function Name: {hello_world.__name__}")
```

```
Function Name: hello_world
```

Timing decorator in action

```
@instrument
def lazy_work(x,y, sleep=2):
    """Sleeps then works"""

    time.sleep(sleep)
    return x+y
```

```
In [7]: lazy_work(4,9)
function: lazy_work, args: [(4, 9), {'sleep': 3}] took: 3.000096082687378 sec
Out[7]: 13
```

Putting it all together

- Functions are units of work
- Functions are more powerful when they take input
- Decorators are functions too

Examples of common decorators

- Many automation tasks involve functions and decorators
  - `flask` web framework
  - `click` command line tool framework
  - `numba` open source JIT compiler
  - custom profiling, tracing and timing

Let's practice!

COMMAND LINE AUTOMATION IN PYTHON

Understand script input

COMMAND LINE AUTOMATION IN PYTHON

Noah Gift

Lecturer, Northwestern & UC Davis & UC Berkeley | Founder, Pragmatic AI Labs

datacamp

Use sys.argv

- `sys.argv` captures input to script

```
import sys
print(sys.argv)
```

```
$ python ./script.py one two
```

```
# Remember that Python lists index at 0
['script.py', 'one', 'two']
```

Parsing arguments

- Parse arguments by index

```
# grab first argument
arg1 = out[1]
print(arg1)
```

```
one
```

Writing a script with sys.argv

- anatomy of a python `sys.argv` script

```
import sys

def hello(user_input):
    print(f"From a user: {user_input}")

if __name__ == "__main__":
    arg1 = sys.argv[1]
    hello(arg1)
```

Parsing input from script

```
python hello_argv.py something
```

```
From a user: something
```

```
python hello_argv.py another
```

```
From a user: another
```

Let's practice!

COMMAND LINE AUTOMATION IN PYTHON

Introduction to Click

COMMAND LINE AUTOMATION IN PYTHON

Noah Gift

Lecturer, Northwestern & UC Davis & UC Berkeley | Founder, Pragmatic AI Labs

datacamp

What is click?

- Python package for creating 'beautiful' command line interfaces
- Three main features:
  - arbitrary nesting of commands
  - automatic help page generation
  - lazy loading of subcommands at runtime

Basic click structure

- import click library

```
import click
```

- use command decorator and option decorator

```
@click.command()
@click.option()
def func(): pass
```

- call function when run as script

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

Simple click example

```
import click


@click.command()
@click.option('--phrase', prompt='Enter a phrase',
              help='')
def tokenize(phrase):
    """tokenize phrase"""

    click.echo(f"tokenized phrase: {phrase.split()}")

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

Using a click application

- running from the terminal

```
python hello_click.py
```

- user prompted to enter phrase

```
Enter a phrase: this is a rabbit
```

- output of click application

```
tokenized phrase: ['this', 'is', 'a', 'rabbit']
```

Automatic help generation

```
? python hello_click.py --help
```

```
Usage: hello_click.py [OPTIONS]

  tokenize phrase

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

Let's practice!

COMMAND LINE AUTOMATION IN PYTHON

Using click to write command line tools

COMMAND LINE AUTOMATION IN PYTHON

Noah Gift

Lecturer, Northwestern & UC Davis & UC Berkeley | Founder, Pragmatic AI Labs

datacamp

Mapping functions to subcommands

```
import click

@click.group()
def cli():
    pass

@cli.command()
def one():
    click.echo('One-1')

@cli.command()
def two():
    click.echo('Two-2')

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

Using click subcommands

```
python click_functions.py
```

```
Usage: click_functions.py [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  one
  two
```

```
python click_functions.py one
```

Click utilities

- `click` utilities can:
  - generate colored output
  - generate paginated output
  - clear the screen
  - wait for key press
  - launch editors
  - write files

```
# Write with click
with click.open_file(filename, 'w') as f:
    f.write('jazz flute')
```

Click and stdout

```
import click
click.echo('Hello DataCamp!')
```

```
Hello DataCamp!
```

- click.echo can:
  - generate colored output
  - generate blinking or bold text
  - print both unicode and binary data

Testing click applications

```
import click
from click.testing import CliRunner
```

```
@click.command()
@click.argument('phrase')
def echo_phrase(phrase):
    click.echo('You said: %s' % phrase)
```

```
runner = CliRunner()
result = runner.invoke(echo_phrase, ['Have data will camp'])
assert result.output == 'You said: Have data will camp\n'
```

Let's practice!

COMMAND LINE AUTOMATION IN PYTHON

Course Summary: Command Line Automation

COMMAND LINE AUTOMATION IN PYTHON

Noah Gift

Lecturer, Northwestern & UC Davis & UC Berkeley | Founder, Pragmatic AI Labs

datacamp

Chapter 1 - IPython and Python interpreter

- Python interpreter
- IPython Shell
- SList data types

Chapter 2 - Shell commands with subprocess

- Execute shell commands
- Using subprocess
- Sending input
- Passing arguments safely

Chapter 3 - Walking the file system

- Dealing with file systems
- Find files matching a pattern
- High-level file and directory operations
- Using pathlib

Chapter 4 - Command line functions

- Using functions for automation
- Understand script input
- Introduction to `click`
- Using `click`

Next steps

COMMAND LINE AUTOMATION IN PYTHON