SEO Tech Developer Residency Week 3: Re-visit Unit Testing/Check webpages

July 12th, 2022

Presented by:
Dr. Sonia Mitchell

SEO Seizing Every Opportunity

Welcome

Office Hours

Mondays, Wednesdays, Thursdays & Fridays

9a.m. - 10:00a.m. EST

Email: Sonia.Mitchell@seo-usa.org

SEO Seizing Every Opportunity

Learning Objectives

At the end of this lesson, you should be able to:

- Write tests to check if webpages exit
- Write tests that check form submissions

Question of Thought

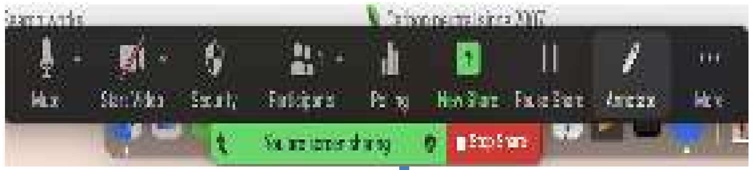
What are do you think you're able to test in your web application so far?

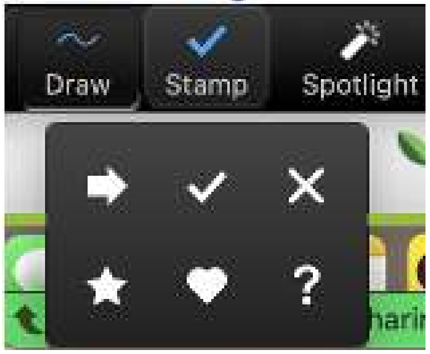
Drop your guesses in the chat!
Or
Unmute Quickly, begin to share

2.0 Setup the test.py structure and run your first test

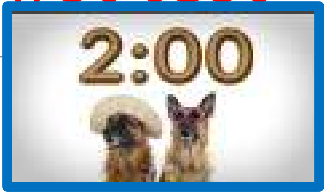
Create a `tests` folder to hold all your tests in the project root.

Add a check stamp below to share that you're done! You can access the stamp tool by clicking *Annotations* in the top Zoom toolbar, then selecting a stamp.

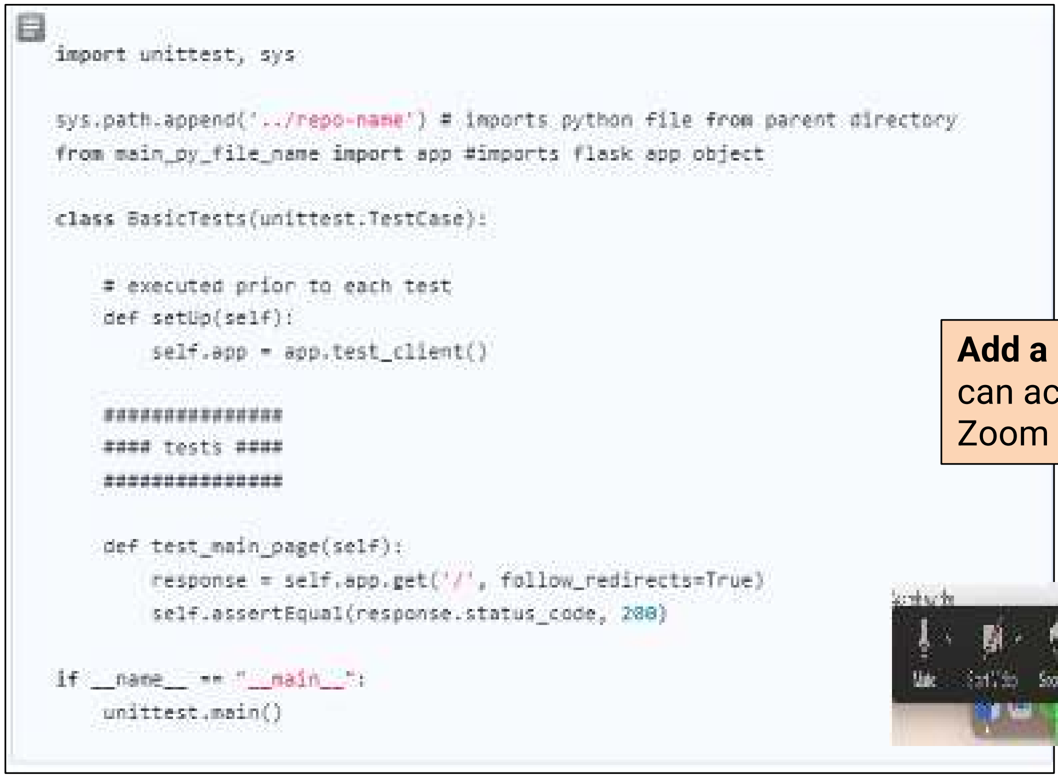




2.0 Setup the test.py structure and run your first test



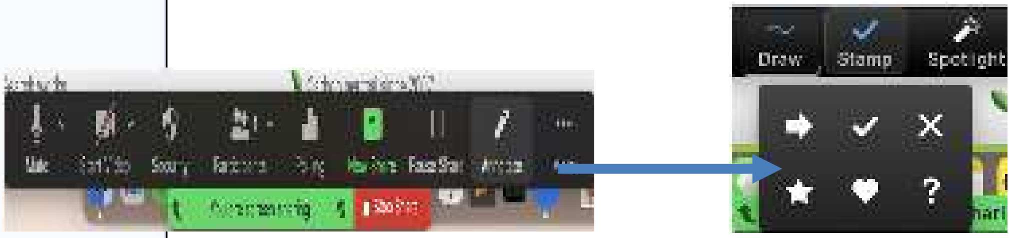
Create a `test_basic.py` file with the following contents:



```
import unittest, sys

sys.path.append('../repo-name') # imports python file from parent directory
from main_py_file_name import app #imports flask app object

class BasicTests(unittest.TestCase):

    # executed prior to each test
    def setUp(self):
        self.app = app.test_client()

    ################
    #### tests ####
    ################

    def test_main_page(self):
        response = self.app.get('/', follow_redirects=True)
        self.assertEqual(response.status_code, 200)

if __name__ == "__main__":
    unittest.main()
```

The test basically does a **GET** request for the webpage and then checks that the status of the request is **200** – which we learned from Week 1 is a successful response.

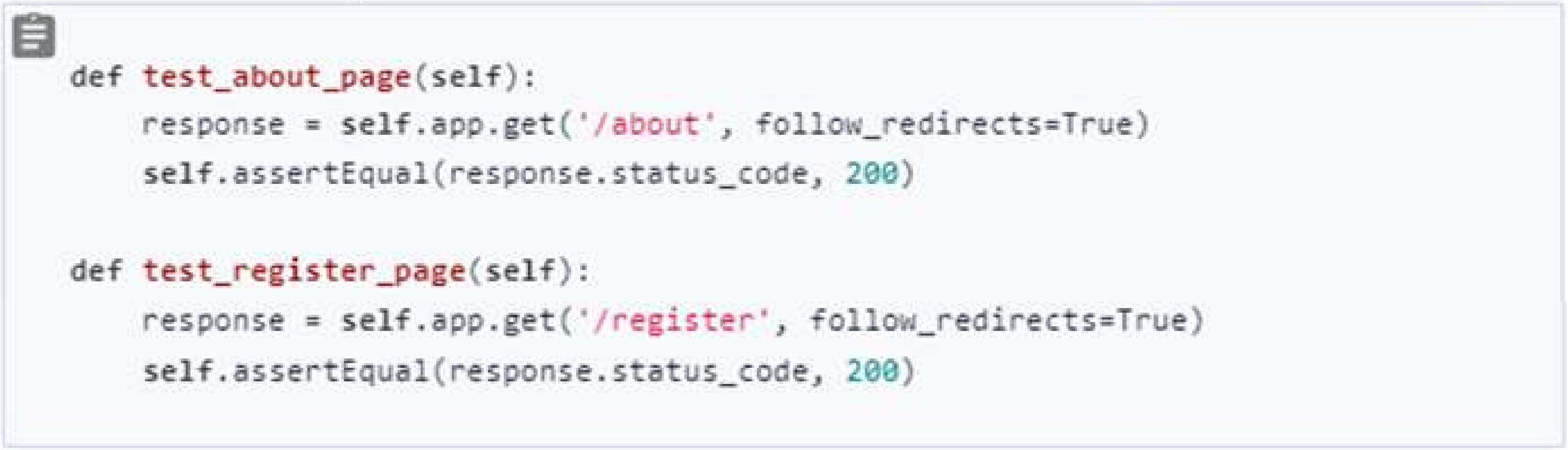
Add a check stamp below to share that you're done! You can access the stamp tool by clicking *Annotations* in the top Zoom toolbar, then selecting a stamp.



2.1 Add tests for other pages

Run your test on the command line from the repo directory `python3tests/test_basic.py`

For every page you have, test that it is live. For example:



```
def test_about_page(self):
    response = self.app.get('/about', follow_redirects=True)
    self.assertEqual(response.status_code, 200)

def test_register_page(self):
    response = self.app.get('/register', follow_redirects=True)
    self.assertEqual(response.status_code, 200)
```

3.0 Set-up Github Actions

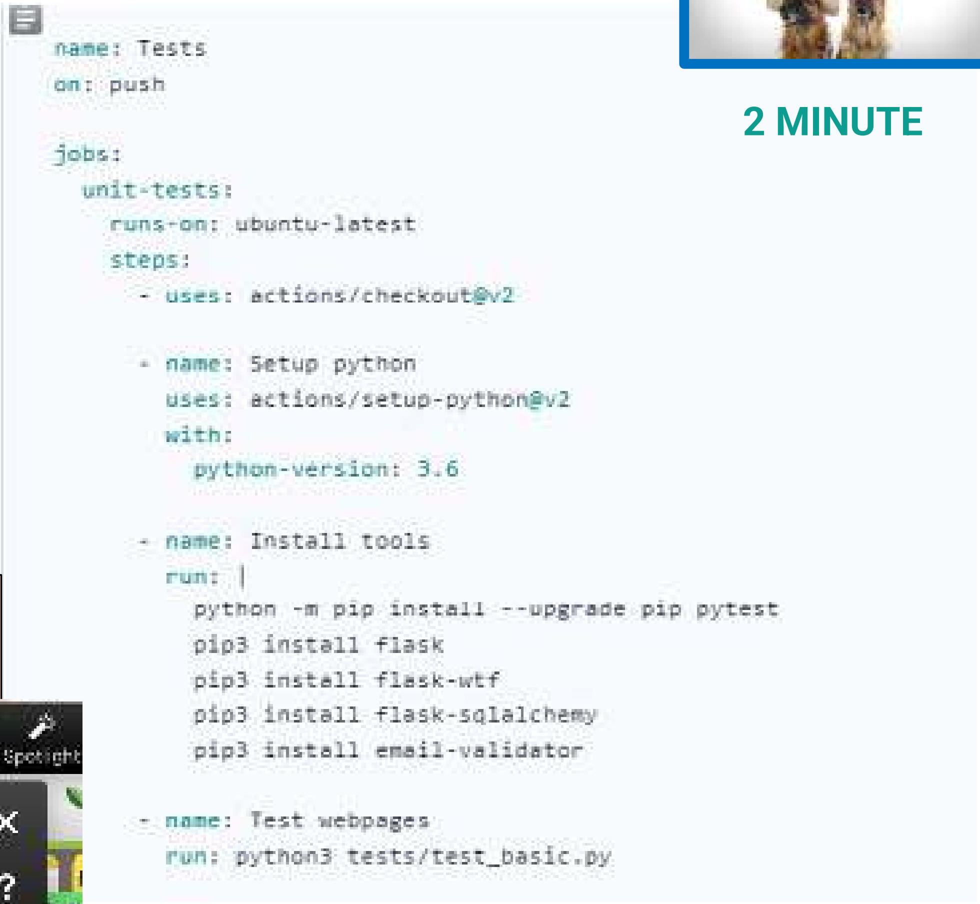
2 MINUTE

Add a `test.yaml` file in

`.github/workflows:`

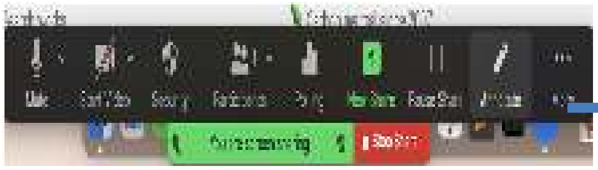
Most is the same – we are just installing more libraries.

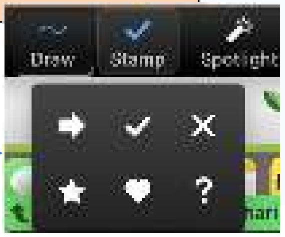
If you push to Github, you should be able to confirm your Github Action is working



```yaml
name: Tests
on: push

jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup python
        uses: actions/setup-python@v2
        with:
          python-version: 3.6

      - name: Install tools
        run: |
          python -m pip install --upgrade pip pytest
          pip3 install flask
          pip3 install flask-wtf
          pip3 install flask-sqlalchemy
          pip3 install email-validator

      - name: Test webpages
        run: python3 tests/test_basic.py
```

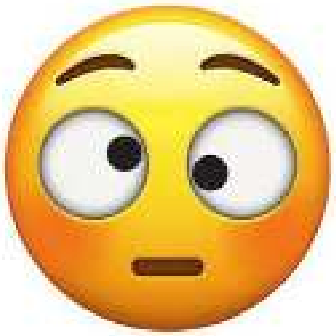
Add a check stamp below to share that you're done! You can access the stamp tool by clicking *Annotations* in the top Zoom toolbar, then selecting a stamp.

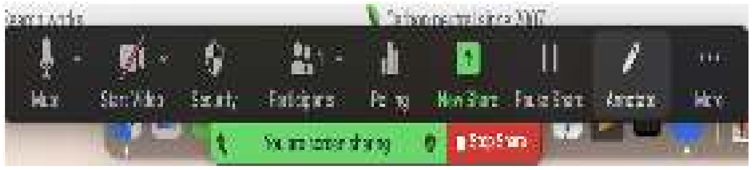


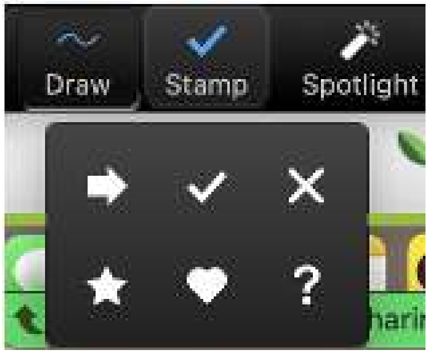


LEARNING TEMPERATURE CHECK

Add a stamp to how you are feeling about the lesson so far. You can access the stamp tool by clicking *Annotations* in the top Zoom toolbar, then selecting a stamp.



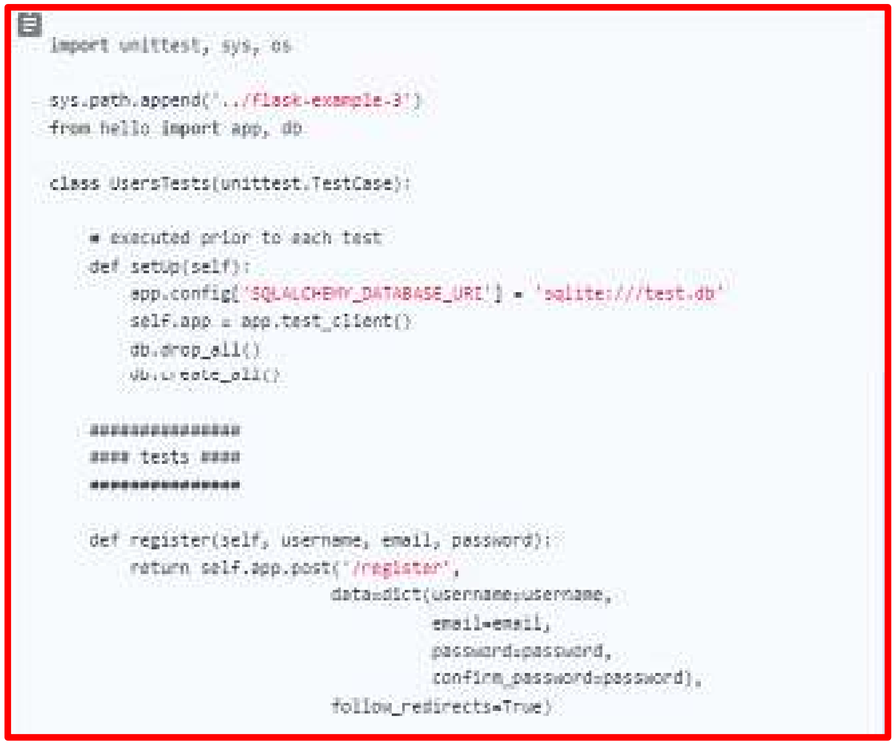




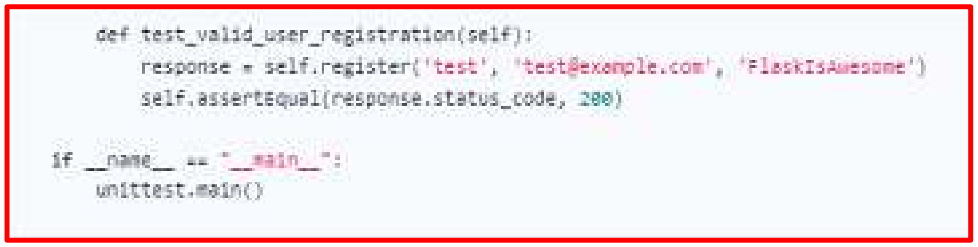
4.0 Testing the Registration form

2 MINUTE

Create a new file `tests` called `test_users.py`



```
import unittest, sys, os

sys.path.append('../flask-example-3')
from hello import app, db

class UsersTests(unittest.TestCase):

    # executed prior to each test
    def setUp(self):
        app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
        self.app = app.test_client()
        db.drop_all()
        db.create_all()

    ###############
    #### tests ####
    ###############

    def register(self, username, email, password):
        return self.app.post('/register',
                             data=dict(username=username,
                                       email=email,
                                       password=password,
                                       confirm_password=password),
                             follow_redirects=True)
```



```
    def test_valid_user_registration(self):
        response = self.register('test', 'test@example.com', 'FlaskIsAwesome')
        self.assertEqual(response.status_code, 200)

if __name__ == "__main__":
    unittest.main()
```

You'll notice there is more setup to handle the database:

- set the database to a new file so we don't overwrite our real data
- drop any data from previous tests
- create the table to get ready for the test

4.1 Testing Invalid Input

To make our lives easier, there is a helper method called register that we can use to test form submissions that returns the response.

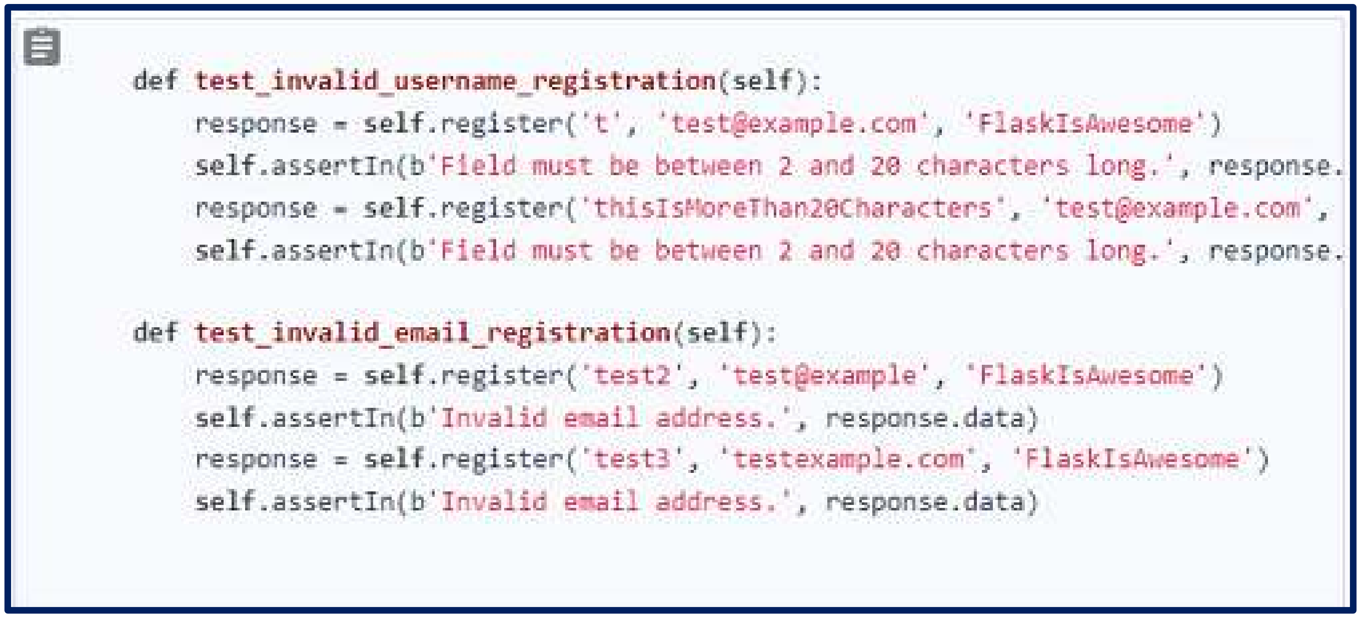
We make use of this in test_valid_user – where we send a valid set of information.

You can now run the test on the command line from the repo directory: `python3 tests/test_users.py`

4.1 Testing Invalid Input

You should also test that your data validation is working: `python3 tests/test_users.py`

Make sure to add a call to your new test file in your : `github/workflows/test.yaml` file

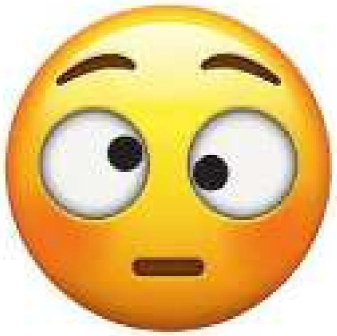


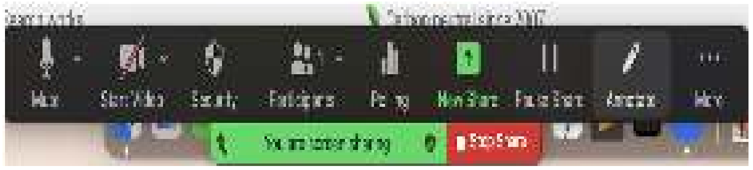
```
def test_invalid_username_registration(self):
    response = self.register('t', 'test@example.com', 'FlaskIsAwesome')
    self.assertIn(b'Field must be between 2 and 20 characters long.', response.
    response = self.register('thisIsMoreThan20Characters', 'test@example.com',
    self.assertIn(b'Field must be between 2 and 20 characters long.', response.

def test_invalid_email_registration(self):
    response = self.register('test2', 'test@example', 'FlaskIsAwesome')
    self.assertIn(b'Invalid email address.', response.data)
    response = self.register('test3', 'testexample.com', 'FlaskIsAwesome')
    self.assertIn(b'Invalid email address.', response.data)
```

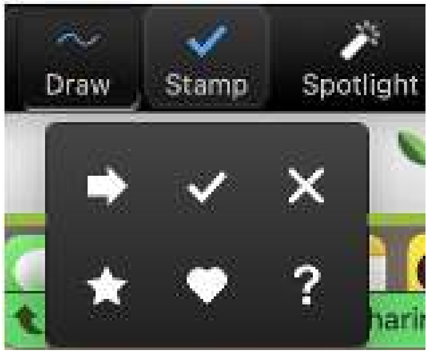
2 MINUTE

LEARNING TEMPERATURE CHECK

Add a stamp to how you are feeling about the lesson so far. You can access the stamp tool by clicking *Annotations* in the top Zoom toolbar, then selecting a stamp.







Did we meet our Learning Objectives(LOs)?

Students **W**ill **B**e **A**ble **T**o

- Write tests to check if webpages exit
- Write tests that check form submissions

Drop your answers in the chat!

Thank You! Q&A

- *SEO Lead Software Engineer Instructor*
  Dr. Sonia Mitchell

Office Hours
Mondays, Wednesdays, Thursdays & Fridays
9a.m. - 10:00a.m. EST
Email: Sonia.Mitchell@seo-usa.org

SEO Seizing Every Opportunity