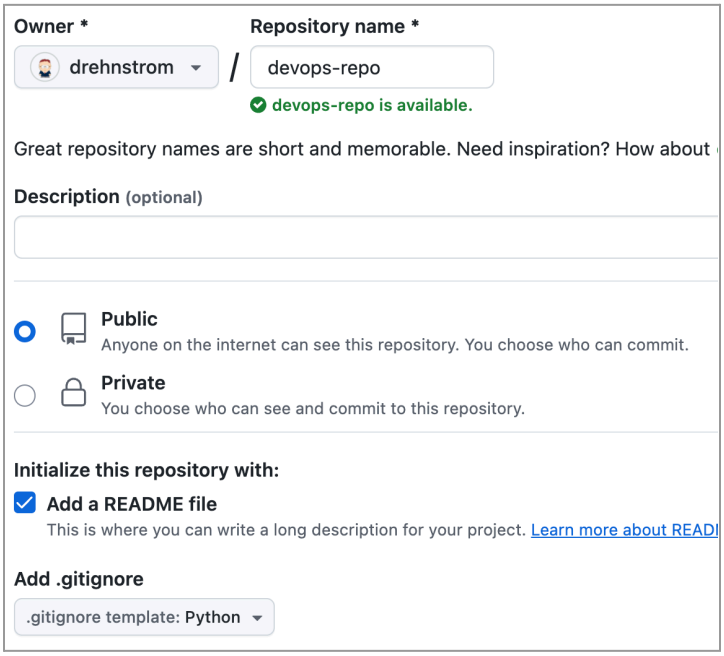
Session 1 Lab: Creating DevOps Pipelines with GitHub Actions

Creating a GitHub Repository

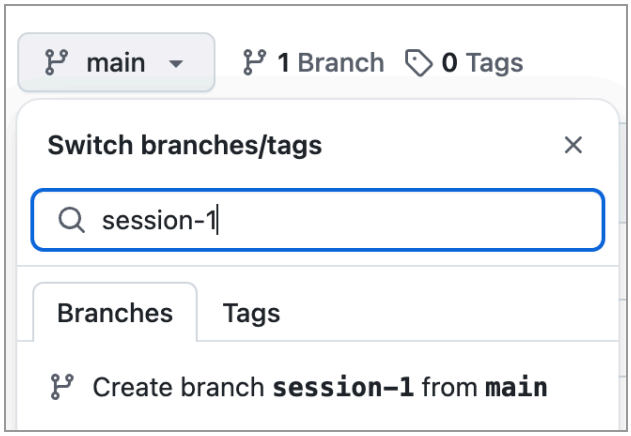
1. Go to https://github.com and click the **Sign in** button. If you already have a GitHub account, log in. If you don't have an account, click the **Create an account** link and follow the instructions.

2. Click the **New** button to create a new repository.

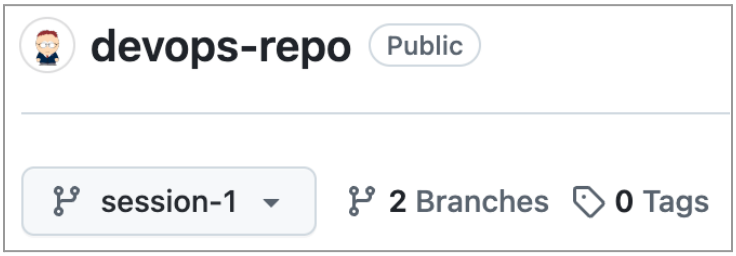
3. Name the repository anything you like. Make the repo **Public**, add a **Readme** file, and select **Python** from the **Add .gitingnore** drop-down.



4. We will do our coding in a branch. Click the dropdown button that reads **main**. Type `session-1` for the branch name and click **Create branch…**.

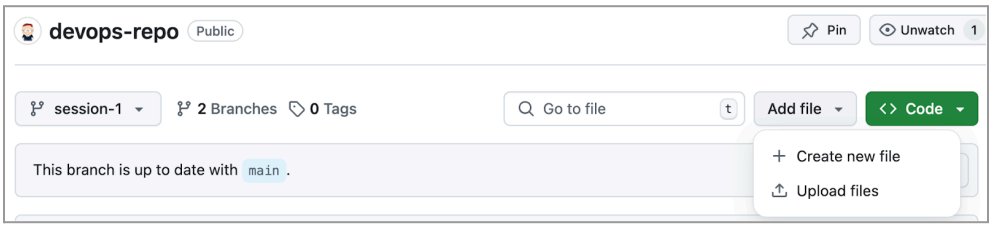


Note: You will automatically switch to the new branch. Make sure you name the branch `session-1,` as this name will be used in our GitHub Actions later.



Adding Code to the Repository

1. From the repo's code page, click the **Add file** or **+** sign button and create a new file.



2. Name the file `app.py` and paste the following code into it. Click the **Commit changes** button to save the file.

```python
from flask import Flask, request
from flask_restful import Resource, Api
```

```
app = Flask(__name__)
api = Api(app)

class Home(Resource):
    def get(self):
        return {'version': '1.0'}

class ConvertTemp(Resource):
    def get(self):
        # Get arguments from query parameters
        temp = float(request.args.get('temp'))
        scale = request.args.get('scale').lower()
        target_scale = request.args.get('target_scale').lower()

        # Perform temperature conversion
        converted_temp = self.convert_temperature(temp, scale, target_scale)

        if converted_temp is None:
            return {'error': 'Invalid scale or target scale'}, 400

        return {'converted_temp': converted_temp, 'target_scale': target_scale}

    def convert_temperature(self, temp, scale, target_scale):
        # Conversion logic
        if scale == target_scale:
            return temp

        if scale == 'celsius':
            if target_scale == 'fahrenheit':
                return temp * 9/5 + 32
            elif target_scale == 'kelvin':
                return temp + 273.15

        elif scale == 'fahrenheit':
            if target_scale == 'celsius':
                return (temp - 32) * 5/9
            elif target_scale == 'kelvin':
                return (temp - 32) * 5/9 + 273.15

        elif scale == 'kelvin':
            if target_scale == 'celsius':
                return temp - 273.15
            elif target_scale == 'fahrenheit':
                return (temp - 273.15) * 9/5 + 32
```

```
        # If scales are invalid
        return None

api.add_resource(Home, '/')
api.add_resource(ConvertTemp, '/convert-temp')

if __name__ == '__main__':
    app.run(debug=True)
```

Note: This may look like a lot of code, but it just creates a simple API using Python Flask that converts temperatures into different scales (Fahrenheit, Celsius, and Kelvin).

3. A Python program uses a requirements file to specify its prerequisites. Add another file called `requirements.txt`, paste the following code into it, and commit the changes.

```
Python
Flask==3.0.3
Flask-RESTful==0.3.10
pytest==8.3.2
```

4. We need to test our code. This can be done with automated unit tests using the Python library called **PyTest**. As you just did, add another file called `test_app.py`. Paste the following code and commit the changes.

```
Python

import pytest
from app import app

@pytest.fixture
def client():
    with app.test_client() as client:
        yield client
```

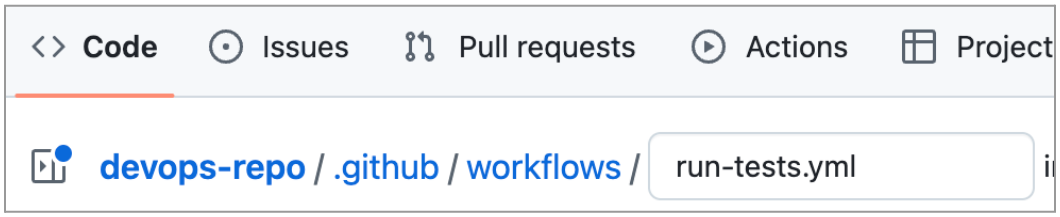
```python
# Home route test
def test_home(client):
    response = client.get('/')
    assert response.status_code == 200
    assert response.json == {'version': '1.0'}

# Fahrenheit to Celsius
@pytest.mark.parametrize("temp, expected", [(212, 100.0), (32, 0.0), (-40,
-40.0)])
def test_convert_temp_f_to_c(client, temp, expected):
    response =
client.get(f'/convert-temp?temp={temp}&scale=fahrenheit&target_scale=celsius')
    assert response.status_code == 200
    assert response.json['converted_temp'] == pytest.approx(expected, rel=1e-2)

# Celsius to Fahrenheit
@pytest.mark.parametrize("temp, expected", [(100, 212.0), (0, 32.0), (-40,
-40.0)])
def test_convert_temp_c_to_f(client, temp, expected):
    response =
client.get(f'/convert-temp?temp={temp}&scale=celsius&target_scale=fahrenheit')
    assert response.status_code == 200
    assert response.json['converted_temp'] == pytest.approx(expected, rel=1e-2)

# Kelvin to Celsius
@pytest.mark.parametrize("temp, expected", [(273.15, 0.0), (373.15, 100.0),
(233.15, -40.0)])
def test_convert_temp_k_to_c(client, temp, expected):
    response =
client.get(f'/convert-temp?temp={temp}&scale=kelvin&target_scale=celsius')
    assert response.status_code == 200
    assert response.json['converted_temp'] == pytest.approx(expected, rel=1e-2)

# Kelvin to Fahrenheit
@pytest.mark.parametrize("temp, expected", [(273.15, 32.0), (373.15, 212.0),
(233.15, -40.0)])
def test_convert_temp_k_to_f(client, temp, expected):
    response =
client.get(f'/convert-temp?temp={temp}&scale=kelvin&target_scale=fahrenheit')
    assert response.status_code == 200
    assert response.json['converted_temp'] == pytest.approx(expected, rel=1e-2)

# Invalid scale handling
def test_convert_temp_invalid_scale(client):
```

```
    response =
client.get('/convert-temp?temp=100&scale=unknown&target_scale=celsius')
    assert response.status_code == 400
    assert 'error' in response.json
```

Using GitHub Actions to Run Automated Tests

1. GitHub Actions are added to Workflows and put into a folder called `.github/workflows`.

   Create another file from your repository code page named `` `.github/workflows/run-tests.yml` ``.



2. Paste the following code into the file, and commit the change.

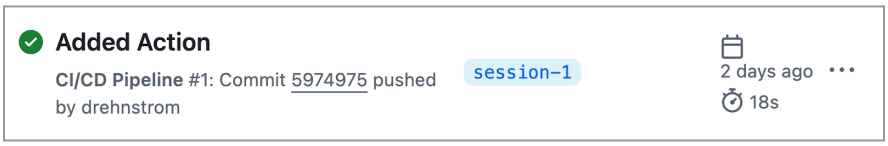
```Python
name: Run Tests on Push to session-1 and Pull Request to main

on:
  push:
    branches:
      - session-1
  pull_request:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
```

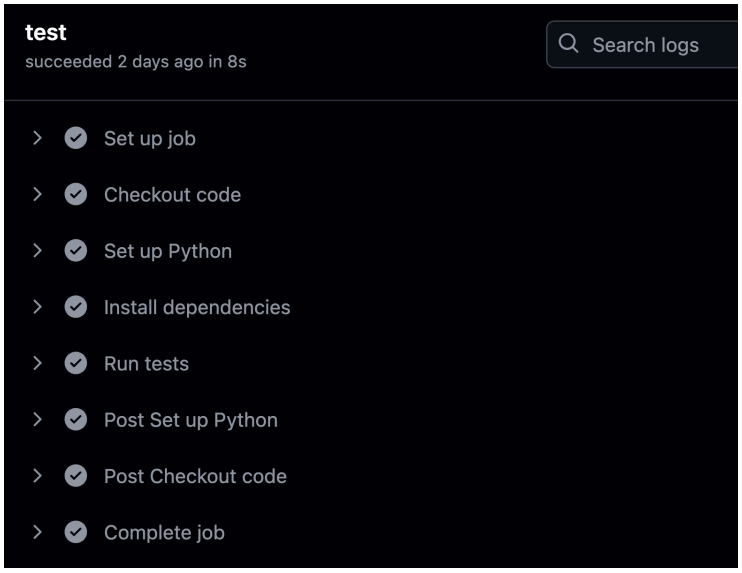
```
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: '3.11'  # Specify the Python version you want to use

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pytest --maxfail=1 --disable-warnings
```

Note: In the example above, the workflow runs when we push to the `session-1` branch or make a pull request to the `main` branch.

There is one job called **test** which has four steps. The first two use the built-in `checkout` and `setup-python` GitHub actions. The third step installs the Python prerequisites using Python pip. That last step runs the tests. These last two steps run the same commands you would run on your computer if you wanted to run the tests manually.

3. Commit the changes, then click the **Actions** link at the top of the page. You should see your workflow run. Click the workflow to see its details.

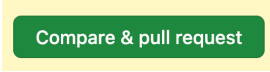




4. Try making a pull request to the main branch. As before, you should see the workflow run. To create a Pull request, click on the Pull Requests menu at the top.

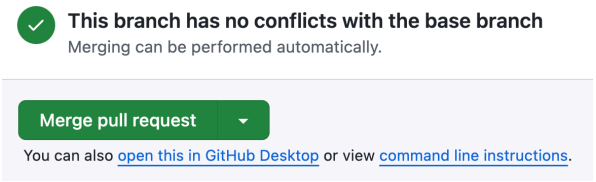
   Then, click the **Compare & pull request** button.

   Add the title `Merge Session 1 to Main` and click **Create pull request**.

   Lastly, go to the Pull Request and click the **Merge pull request** button.

5. Experiment a little. Break the code and commit it so you can see the tests fail, then fix the code and make sure the tests pass.

6. Make sure when you are done making experiments you create a pull request from the session 1 branch to the main branch and merge your changes.