Part 2: Lecture 1

TECH2: Introduction to Programming, Data, and Information Technology

Richard Foltyn

Norwegian School of Economics (NHH)

September 27, 2024

Course contents

Agenda for Part 2 of the course

1. Working with standard development tools
2. Working with data using pandas

Same approach as in Part 1:

- (hands-on) lectures on Wednesday
- Workshops on Friday

Contents of lecture 1

Version control with Git

Git

Why git? (and GitHub)

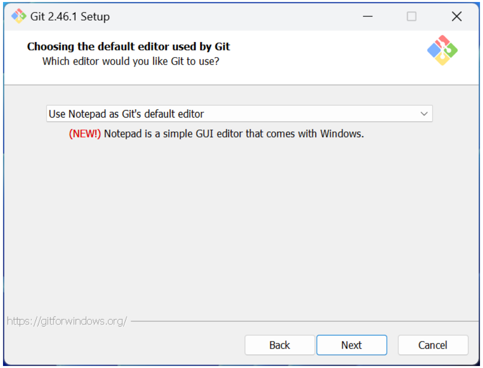
- Because everyone uses it: almost completely wiped out other version control systems over the last 19 years

  Examples:
  - Python: https://github.com/python/cpython
  - NumPy: https://github.com/numpy/numpy
  - SciPy: https://github.com/scipy/scipy
  - Pandas: https://github.com/pandas-dev/pandas
  - Matplotlib: https://github.com/matplotlib/matplotlib
  - PyTorch (Meta's ML library): https://github.com/pytorch/pytorch
  - TensorFlow (Google's ML library): https://github.com/tensorflow/tensorflow
- Keeps history of **your** code changes (and restore previous versions)
- Keeps history of **other's** code changes
- Allows for decentralized coding in teams
- Allows syncing of code across devices

Windows

- Download from https://git-scm.com/download/win
- The installer allows you to tweak various things, just accept the defaults.
- One setting you might want to change is the default editor for commit messages.

  Sensible choices: Notepad (graphical editor), Nano (command line)


Git 2.46.1 Setup

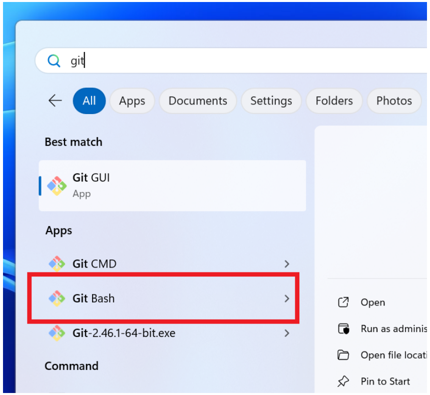
Choosing the default editor used by Git
Which editor would you like Git to use?

Use Notepad as Git's default editor

(NEW!) Notepad is a simple GUI editor that comes with Windows.

https://gitforwindows.org/

- Once installed, git can be used via the command line by launching "Git Bash"



- **Do not** use the variant that is called “Git CMD” as this will open a different type of terminal.

Git

Installation — macOS & Linux

macOS

- Git for macOS cannot be downloaded from Git website directly
- Alternatives:
  1. Install from Apple Developer Command Line Tools:

     In the terminal, type

     ```
     xcode-select --install
     ```

     and follow the instructions.
     See https://mac.install.guide/commandlinetools/4 for details.
  2. Install from `homebrew`: https://youtu.be/B4qsvQ5IqWk
- Once installed, you can use git directly from your regular Terminal application

Linux

- Install git directly from your distribution's package repository

Git
Configuration

- Before using git, you need to set your name and email address
- Open the terminal (Git Bash on Windows, Terminal on macOS) and type the following (use your name and NHH email address):

```
git config --global user.name "John Doe"
git config --global user.email johndoe@student.nhh.no
```

Best practices

- Choose a base folder for all your repositories (e.g., `repos/` in your user directory)
- Never put your git repository into a directory that is synced with the web (iCloud, DropBox, Google Drive, OneDrive, etc.)
- Learn to use git on the command line before moving to a graphical user interface (GUI)
- Try to make smaller, meaningful, self-contained commits instead of dumping days' worth of work into a single commit
- Don't put large binary files into git (such as PDFs, images, videos, MS Office documents). Version control is for text files!

Typical git workflow

1. Initialize new repository (needed only once per project):
   Change to the desired project directory and run:

   ```
   git init -b main          # Optionally give a name to main brach
   ```

2. Edit your source code, add new files, etc.
3. Check what changed:

   ```
   git status
   git diff
   ```

4. Add changed files to git:

   ```
   git add file.py       # add specific file
   git add .             # add all files and sub-folders in current folder
   ```

5. Create a new commit:

   ```
   git commit -m "Commit message"      # The commit message is optional
   ```

6. Push changes to a remote repository (if you have one):

   ```
   git push
   ```

Example 1

Example 1 — Your turn!

In this example, we implement a function `argmax()` which returns the position and value of a maximum in a given sequence.

The example will take you through the following steps:

1. Create a folder for your git repository, e.g., `tech2-part2-lecture1`
2. Initialize the git repository
3. Create the file `example1.py`
4. Add initial version of `example1.py` to git
5. Commit initial version
6. Implement the function `argmax()`
7. Inspect the changes
8. Add modified `example1.py` to git
9. Commit updated version

Consult the list of terminal commands and git commands you go along!

Example 1 — Step 1: Create a folder

- You should create a **dedicated** folder that will **exclusively** store your project files
- Avoid using existing general folders such as your home folder, `Documents`, `Desktop`
- You can create a folder using Windows Explorer, Finder on macOS, or do it via the command line (Terminal on macOS, Git Bash on Windows)
- To create a folder in your home directory, run the following commands:

```
cd
mkdir tech2-part2-lecture1
cd tech2-part2-lecture1
```

  - The command `cd` ("**c**hange **d**irectory") is used to change the current working directory
  - Without an argument, `cd` changes to your home folder
  - The command `mkdir` ("**make dir**ectory") creates a new directory
- Avoid using spaces in folder or file names. With spaces, you need to wrap names in quotes, e.g.,

```
cd "Tech 2 - Part 2 - Lecture 1"
```

See terminal commands

Example 1 — Step 2: Initialize the git repository

- Before adding any files to git, you need to initialize a folder as a git repository
- Assuming you are in the folder `tech2-part2-lecture1`, run

  ```
  git init -b main
  ```

- This command might fail with older versions of git (on older macOS systems). In that case, run

  ```
  git init
  git branch -M main
  ```

- You can confirm that the repository was initialized using

  ```
  git status
  ```

  which should produce the output

  ```
  No commits yet

  nothing to commit (create/copy files and use "git add" to track)
  ```

Example 1 — Step 3: Create the file `example1.py`

- Create the file `example1.py` and store it in the git repository folder
- You can use the template file provided on Github using this link
- Run

  ```
  git status
  ```

  to confirm that `example1.py` is present:

  ```
  No commits yet

  Untracked files:
    (use "git add <file>..." to include in what will be committed)
          example1.py

  nothing added to commit but untracked files present (use "git add" to track)
  ```

  This tells you that the file `example1.py` is present, but currently not tracked by git.

Example 1 — Step 4: Add `example1.py` to git

- Before we can commit it, we need to add `example1.py` to git
- We do this by running

```
git add example1.py
```

- To confirm that the file was added, run

```
git status
```

```
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   example1.py
```

This tells you that the file `example1.py` is is now ready to be committed.

Example 1 — Step 5: Commit the initial version

- You are now ready to commit the initial version of `example1.py` to git
- We do this by running

  ```
  git commit -m "Initial commit"
  ```

- Alternatively, you can run

  ```
  git commit
  ```

  In that case, an editor will open where you can type a commit message.
- After commit, your repository is clean (no changed files):

  ```
  git status
  ```

  ```
  nothing to commit, working tree clean
  ```

- You can check the commit log to see your commit:

  ```
  git log
  ```

  ```
  commit 0c727bdbb27d2019232e3c2ba9bc0e9ea5699d67 (HEAD -> main)
  Author: Richard Foltyn <richard.foltyn@gmail.com>
  Date:   Wed Sep 25 17:10:37 2024 +0200

      Initial commit
  ```

Example 1 — Step 6: Implement argmax()

Implement the following function using the template code in example1.py:

```
def argmax(values):
    """
    Return the location and value of the maximum contained in a given sequence.

    Parameters
    ----------
    values : Sequence of numbers

    Returns
    -------
    imax : int
        Location of the maximum
    vmax : int or float
        Maximum value
    """
```

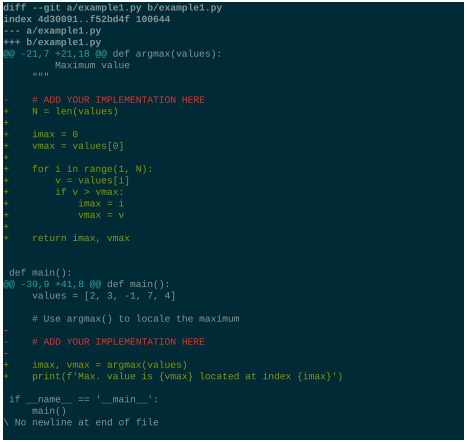
Test your function with the following list of values:

```
values = [2, 3, -1, 7, 4]
```

Example 1 — Step 7: Inspect changes

- Once you have completed your changes to `example1.py`, you can display the changes as follows:

```
git diff
```



```
diff --git a/example1.py b/example1.py
index 4d30091..f52bd4f 100644
--- a/example1.py
+++ b/example1.py
@@ -21,7 +21,18 @@ def argmax(values):
         Maximum value
     """

-    # ADD YOUR IMPLEMENTATION HERE
+    N = len(values)
+
+    imax = 0
+    vmax = values[0]
+
+    for i in range(1, N):
+        v = values[i]
+        if v > vmax:
+            imax = i
+            vmax = v
+
+    return imax, vmax


 def main():
@@ -30,9 +41,8 @@ def main():
     values = [2, 3, -1, 7, 4]

     # Use argmax() to locale the maximum
-
-    # ADD YOUR IMPLEMENTATION HERE
-
+    imax, vmax = argmax(values)
+    print(f'Max. value is {vmax} located at index {imax}')

 if __name__ == '__main__':
     main()
\ No newline at end of file
```

Example 1 — Step 8: Add modified `example1.py` to git

- Running `git status` shows that `example1.py` has been modified:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example1.py

no changes added to commit (use "git add" and/or "git commit -a")
```

- Before we can commit our changes, we **again** need to add `examples1.py` to git.

  This needs to be done each time a file has been changed, not only if new files are added to the repository!

```
git add example1.py
```

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   example1.py
```

Example 1 — Step 9: Commit changes

- We are now ready to commit the changes to `example1.py`:

```
git commit -m "Add implementation of argmax()"
```

```
 1 file changed, 14 insertions(+), 4 deletions(-)
```

- The commit log now shows the new commit:

```
git log
```

```
commit 892eb57fdf25ba9dc3050fea180b3bf3f8181524 (HEAD -> main)
Author: Richard Foltyn <richard.foltyn@gmail.com>
Date:   Wed Sep 25 17:34:34 2024 +0200

    Add implementation of argmax()

commit 0c727bdbb27d2019232e3c2ba9bc0e9ea5699d67
Author: Richard Foltyn <richard.foltyn@gmail.com>
Date:   Wed Sep 25 17:10:37 2024 +0200

    Initial commit
```

Example 1b

Example 1b — Add error handling

- Make sure you have committed your implementation of `argmax()` from Example 1 before proceeding.
- Test your `argmax()` with an empty sequence, e.g., `[]`

  Chances are that it does not correctly handle that case (if it does: well done!)

Your turn:

1. Extend `argmax()` to raise a `ValueError` exception when given an empty sequence.
2. Add code to test that it works, e.g., by calling `argmax()` with an empty list

   ```
   argmax([])
   ```

3. Inspect the differences to your previous commit using

   ```
   git diff
   ```

4. Add the modified file `example1.py` to git
5. Commit your changes

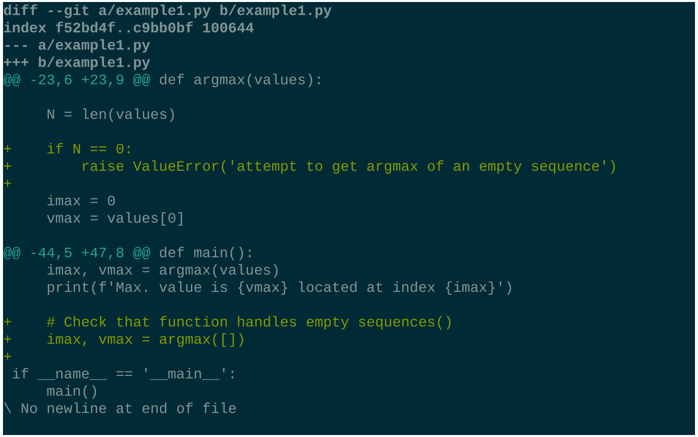
Example 1b — Steps 1 & 2

- If you didn't complete Example 1, you can download the solution here and use it as your starting point.

Example 1b — Step 3

- Once you have updated `example1.py`, use `git diff` to see the changes:

  ```
  git diff
  ```



```
diff --git a/example1.py b/example1.py
index f52bd4f..c9bb0bf 100644
--- a/example1.py
+++ b/example1.py
@@ -23,6 +23,9 @@ def argmax(values):

     N = len(values)

+    if N == 0:
+        raise ValueError('attempt to get argmax of an empty sequence')
+
     imax = 0
     vmax = values[0]

@@ -44,5 +47,8 @@ def main():
     imax, vmax = argmax(values)
     print(f'Max. value is {vmax} located at index {imax}')

+    # Check that function handles empty sequences()
+    imax, vmax = argmax([])
+
 if __name__ == '__main__':
     main()
\ No newline at end of file
```

Example 1b — Steps 4 & 5

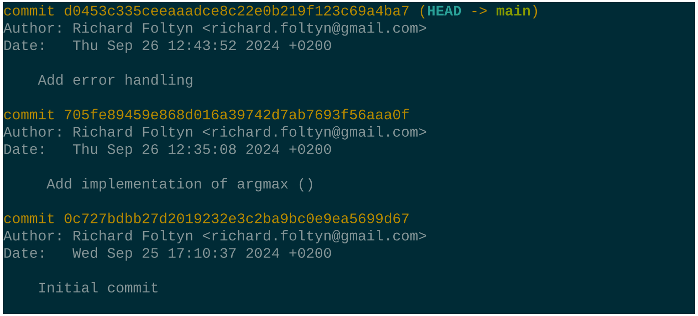
- You can now add your changes to git:

  ```
  git add example1.py
  ```

- …and create a new commit:

  ```
  git commit -m "Add error handling"
  ```

- Running `git log` shows the new commit:



```
commit d0453c335ceeaaadce8c22e0b219f123c69a4ba7 (HEAD -> main)
Author: Richard Foltyn <richard.foltyn@gmail.com>
Date:   Thu Sep 26 12:43:52 2024 +0200

    Add error handling

commit 705fe89459e868d016a39742d7ab7693f56aaa0f
Author: Richard Foltyn <richard.foltyn@gmail.com>
Date:   Thu Sep 26 12:35:08 2024 +0200

     Add implementation of argmax ()

commit 0c727bdbb27d2019232e3c2ba9bc0e9ea5699d67
Author: Richard Foltyn <richard.foltyn@gmail.com>
Date:   Wed Sep 25 17:10:37 2024 +0200

    Initial commit
```

GitHub

GitHub

Why GitHub?

- Everyone uses it!

  Alternatives:

  - GitLab
  - BitBucket
- Offers many other services besides version control (issue tracking, Wiki, etc.)
- Register for free at https://github.com/signup

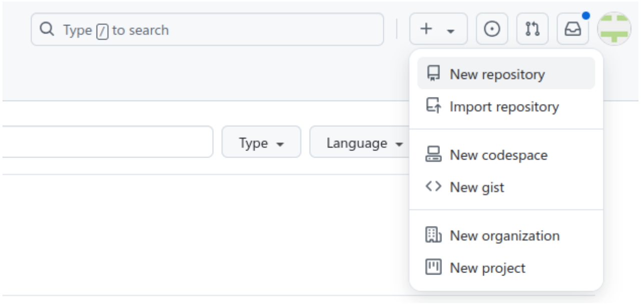
  Use your NHH address!

Local vs. remote repositories

- So far, our git repository was **local** — no way to access it from another machine or share it with others.
- To push the repository to the cloud, we need to
  1. Create a (remote) cloud-hosted repository (on GitHub, etc.)
  2. Link our local repository to the remote repository

Creating a GitHub repository

- Go to GitHub and click in the right corner to create a new repository

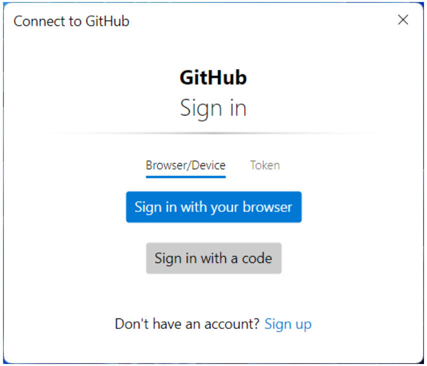


GitHub authentication

- GitHub no longer supports authentication via username + password (only on the website directly)
- Depending on your operating system, you need to set up authentication to work with git on the command line
- See the following slides for Windows and macOS / Linux

GitHub authentication — Windows

- This should work automatically with recent git versions
- The first time you interact with GitHub via git on the command line, you should see the following:



- Select "Sign in with your browser" and follow the instructions
- The authentication token is stored locally so you have to do this only once

GitHub authentication — macOS (& Linux)

- Most likely, GitHub authentication does not work out of the box on macOS
- You have to configure an SSH key that will be used for authentication
- Run the following commands

  ```
  mkdir ~/.ssh
  ssh-keygen  -t ed25519
  ```

  Accept the default file location and choose an empty password
- Print your public SSH key:

  ```
  cat ~/.ssh/id_ed25519
  ```

  This prints something like

  ```
  ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAICrt0hjm1u3m+k3EXIhaBoCEGWFDk4qFaGibEcr17Z+
  ```
- Copy this output and add it to your GitHub account as described here

Linking local to remote repositories

Once the repository is created, GitHub shows you instructions on how to link the newly created repository to your existing local one.

- If you use the browser-based (HTTPS) login method, run

```
git remote add origin https://github.com/richardfoltyn/tech2-part2-lecture1.git
git push -u origin main
```

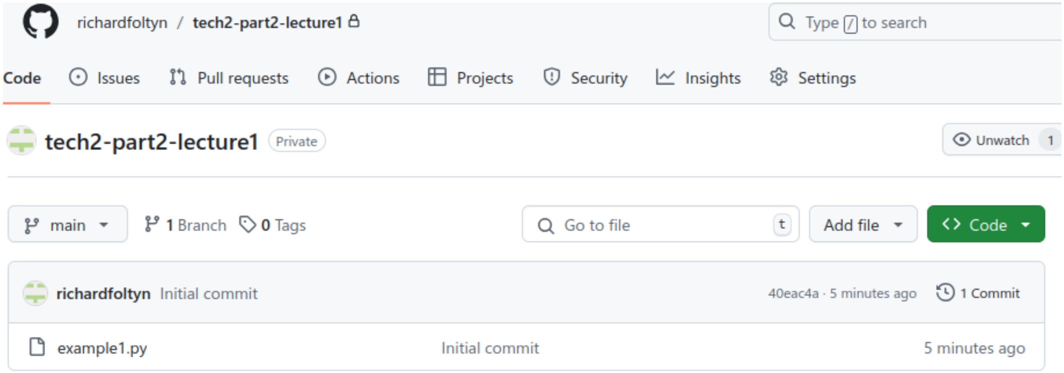
- If you authenticate with SSH keys, run

```
git remote add origin git@github.com:richardfoltyn/tech2-part2-lecture1.git
git push -u origin main
```

You'll of course have to use the URL to **your own** repository.

Remote repository with new content

Once you have pushed your commits, you can browse them in your GitHub repository:



richardfoltyn Initial commit 40eac4a · 5 minutes ago 1 Commit

example1.py	Initial commit	5 minutes ago

Example 2

Example 2: putting `argmax()` to use

- Image you are given 100 skillingsboller (cinnamon buns)
- The utility you derive from consuming c buns a day is represented by the function

$$u(c) = -(c/50 - 1.5)^2 + 2$$

- You can split your consumption of buns across two days, c_1 and c_2, so that

$$c_1 + c_2 = 100, \qquad c_1 \geq 0,\ c_2 \geq 0$$

- Your total utility over two days is given by

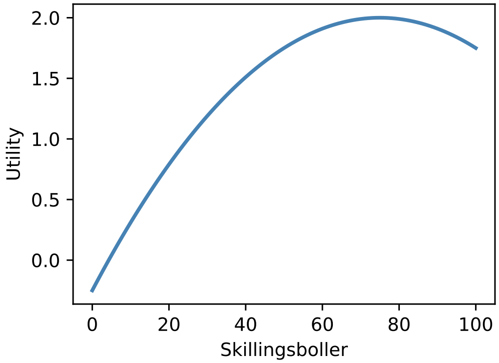
$$U(c_1, c_2) = u(c_1) + u(c_2)$$

- How many buns should you eat each day?

Utility function

The per-day utility from consuming c buns is

$$u(c) = -\left(\frac{c}{50} - 1.5\right)^2 + 2 \tag{1}$$



Example 2 — Your turn!

Create a file `example2.py` and implement the following functions [template]:

```python
def utility(c):
    """
    Return per-day utility of consumption c buns.
    """


def utility_total(c1, c2):
    """
    Return total utility of consuming c1 today and c2 tomorrow.
    """
```

Use the function `argmax()` you wrote earlier to find the optional consumption today. Evaluate 101 candidate consumption levels given by

```python
buns = 100
c1_cand = np.linspace(0, buns, 101)
```

Committing and pushing your changes

- When you are done with your implementation, add your new file to git:

```
git add example2.py
```

- You might have modified `example1.py` as well, then you need to add it:

```
git add example1.py
```

- Create a commit:

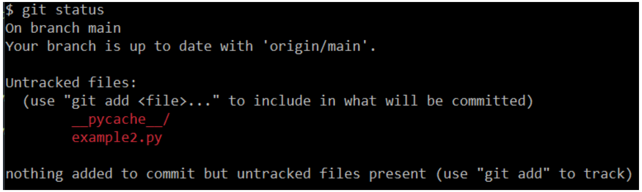
```
git commit -m "Find optimal bun consumption"
```

- Push your commit to GitHub

```
git push
```

Telling git to ignore files

- Depending on how you ran your code, you might notice additional items in your project folder:



```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        __pycache__/
        example2.py

nothing added to commit but untracked files present (use "git add" to track)
```

- We never want such cache and temporary files to end up in our repository
- We can tell git to ignore them by creating a file called `.gitignore` in the root folder of the repository
- Each line in this file specifies a pattern or file name to ignore:

```
__pycache__/
```

Visual Studio Code

Visual Studio Code

Why Visual Studio Code?

- Has become the most widely used editor for most languages (see StackOverflow Developer Survey 2024)
- Free & open source
- Good support for almost any programming language and file format (e.g., Jupyter Notebooks) via extensions
- Natively supports git & GitHub (unlike Spyder and older editors)

- Alternative: PyCharm by JetBrains (free community edition is available, free professional edition for students)

- Note: Visual Studio Code completely independent of Visual Studio, a commercial IDE from Microsoft for Windows development

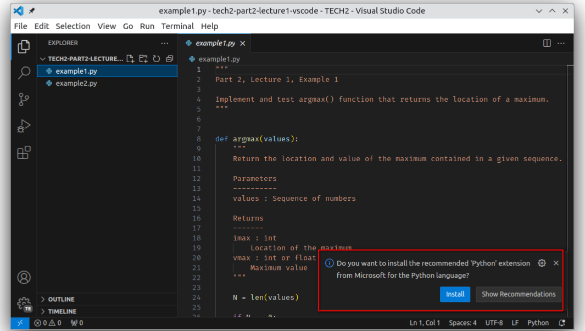
Installation

Visual Studio Code

- Download from project website https://code.visualstudio.com/download
- Installation is straightforward, just confirm defaults
    - Optionally, on Windows you can choose to create an icon on the Desktop, integrate VS Code into the Explorer context menu, etc.
- Python-specific quick start guide:
  https://code.visualstudio.com/docs/python/python-quick-start
- Useful extensions (you'll be promted to install them as you open Python files):
    - Python
    - Jupyter

Using Visual Studio Code

- Create a new directory, say `tech2-part2-lecture1-vscode` and copy over `example1.py` and `example2.py`
- In VS Code, open this directory:



example1.py - tech2-part2-lecture1-vscode - TECH2 - Visual Studio Code

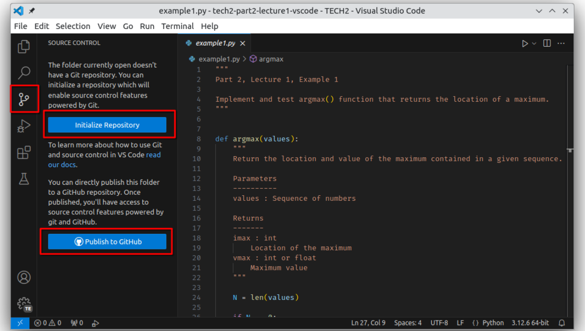
```
"""
Part 2, Lecture 1, Example 1

Implement and test argmax() function that returns the location of a maximum.
"""


def argmax(values):
    """
    Return the location and value of the maximum contained in a given sequence.

    Parameters
    ----------
    values : Sequence of numbers

    Returns
    -------
    imax : int
        Location of the maximum
    vmax : int or float
        Maximum value
    """

    N = len(values)
```

Do you want to install the recommended 'Python' extension from Microsoft for the Python language?

- When you open Python (or other) source files, VS Code will suggest to install extensions to support that file type.

VS Code & git & GitHub

- Support for git & GitHub are fully integrated into VS Code.
- On the "Source Control" tab you can directly create a repository (`git init`) or directly create a GitHub repository:


example1.py - tech2-part2-lecture1-vscode - TECH2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
SOURCE CONTROL
example1.py
example1.py > argmax
Ln 27, Col 9 Spaces: 4 UTF-8 LF {} Python 3.12.6 64-bit

The folder currently open doesn't have a Git repository. You can initialize a repository which will enable source control features powered by Git.

Initialize Repository

To learn more about how to use Git and source control in VS Code read our docs.

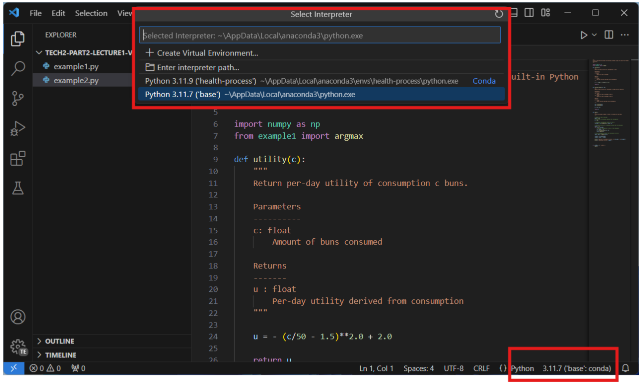
You can directly publish this folder to a GitHub repository. Once published, you'll have access to source control features powered by git and GitHub.

Publish to GitHub

```python
"""
Part 2, Lecture 1, Example 1

Implement and test argmax() function that returns the location of a maximum.
"""


def argmax(values):
    """
    Return the location and value of the maximum contained in a given sequence.

    Parameters
    ----------
    values : Sequence of numbers

    Returns
    -------
    imax : int
        Location of the maximum
    vmax : int or float
        Maximum value
    """

    N = len(values)
```

VS Code: Running Python

- In order to run Python code, you need to select a Python interpreter (or Anaconda environment)
- You can do this in the lower-right corner, which will bring up a list of interpreters that VS Code found on your system.



File Edit Selection View
Select Interpreter
Selected Interpreter: ~\AppData\Local\anaconda3\python.exe
+ Create Virtual Environment...
Enter interpreter path...
Python 3.11.9 ('health-process') ~\AppData\Local\anaconda3\envs\health-process\python.exe Conda
Python 3.11.7 ('base') ~\AppData\Local\anaconda3\python.exe
EXPLORER
TECH2-PART2-LECTURE1-V
example1.py
example2.py
OUTLINE
TIMELINE
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.11.7 ('base': conda)

```
import numpy as np
from example1 import argmax

def utility(c):
    """
    Return per-day utility of consumption c buns.

    Parameters
    ----------
    c: float
        Amount of buns consumed

    Returns
    -------
    u : float
        Per-day utility derived from consumption
    """

    u = - (c/50 - 1.5)**2.0 + 2.0

    return u
```

Example 3

Example 3: Using NumPy's `argmax()`

- We used our own implementation of `argmax()` to locate the utility-maximizing consumption level
- NumPy comes with an implementation of `argmax()` which should be preferred over creating our own

Your turn!

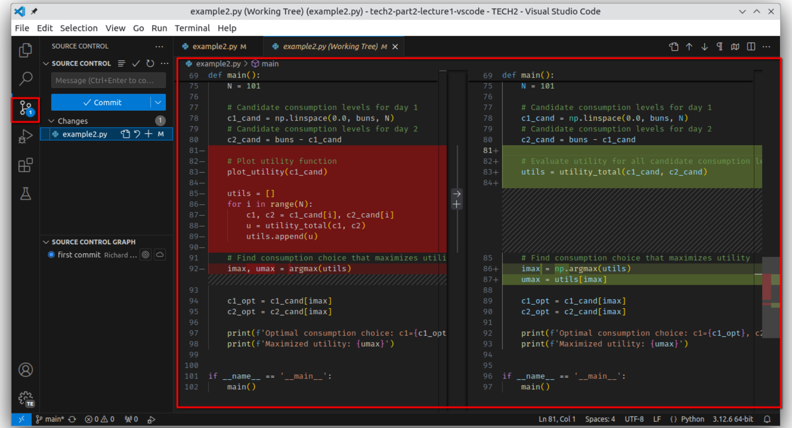
1. Modify your code to use NumPy's `argmax()`.
   Hint: NumPy's version returns only the location, not the maximum value!
2. Use VS Code's "Source Control" tab to show the changes vs. your previous implementation.
3. Add and commit your changes using Visual Studio Code.
4. Push your commits to GitHub.

Example 3 — Step 1

- If you did not complete Example 2, you can download the solution here and use it as a starting point.

Example 3 — Step 2: Show changes in VS Code

The VS Code "Source Control" tab shows changes relative to your last commit:


example2.py (Working Tree) (example2.py) - tech2-part2-lecture1-vscode - TECH2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
SOURCE CONTROL
Message (Ctrl+Enter to co...
✓ Commit
Changes 1
example2.py M
SOURCE CONTROL GRAPH
first commit Richard ...
example2.py M
example2.py (Working Tree) M
example2.py > main
main* 0 0 0
Ln 81, Col 1 Spaces: 4 UTF-8 LF {} Python 3.12.6 64-bit

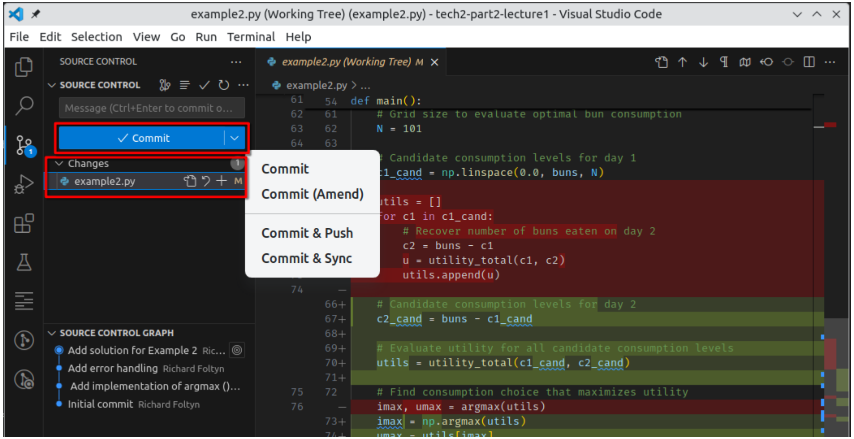
```python
def main():
    N = 101

    # Candidate consumption levels for day 1
    c1_cand = np.linspace(0.0, buns, N)
    # Candidate consumption levels for day 2
    c2_cand = buns - c1_cand

    # Plot utility function
    plot_utility(c1_cand)

    utils = []
    for i in range(N):
        c1, c2 = c1_cand[i], c2_cand[i]
        u = utility_total(c1, c2)
        utils.append(u)

    # Find consumption choice that maximizes utili
    imax, umax = argmax(utils)

    c1_opt = c1_cand[imax]
    c2_opt = c2_cand[imax]

    print(f'Optimal consumption choice: c1={c1_opt
    print(f'Maximized utility: {umax}')


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

```python
def main():
    N = 101

    # Candidate consumption levels for day 1
    c1_cand = np.linspace(0.0, buns, N)
    # Candidate consumption levels for day 2
    c2_cand = buns - c1_cand

    # Evaluate utility for all candidate consumption le
    utils = utility_total(c1_cand, c2_cand)

    # Find consumption choice that maximizes utility
    imax = np.argmax(utils)
    umax = utils[imax]

    c1_opt = c1_cand[imax]
    c2_opt = c2_cand[imax]

    print(f'Optimal consumption choice: c1={c1_opt}, c2
    print(f'Maximized utility: {umax}')


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

Example 3 — Steps 3 & 4: Add and commit changes

The VS Code "Source Control" tab allows you to

1. add changes to git (click on the "+" next to each file) — this is equivalent to `git add`
2. commit changes (click on the "Commit" button) — this is equivalent to `git commit`


example2.py (Working Tree) (example2.py) - tech2-part2-lecture1 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
SOURCE CONTROL
SOURCE CONTROL
Message (Ctrl+Enter to commit o...
✓ Commit
Changes
example2.py M
Commit
Commit (Amend)
Commit & Push
Commit & Sync
SOURCE CONTROL GRAPH
Add solution for Example 2 Ric...
Add error handling Richard Foltyn
Add implementation of argmax ()...
Initial commit Richard Foltyn
example2.py (Working Tree) M
example2.py > ...

```
def main():
    # Grid size to evaluate optimal bun consumption
    N = 101

    # Candidate consumption levels for day 1
    1_cand = np.linspace(0.0, buns, N)

    tils = []
    or c1 in c1_cand:
        # Recover number of buns eaten on day 2
        c2 = buns - c1
        u = utility_total(c1, c2)
        utils.append(u)

    # Candidate consumption levels for day 2
    c2_cand = buns - c1_cand

    # Evaluate utility for all candidate consumption levels
    utils = utility_total(c1_cand, c2_cand)

    # Find consumption choice that maximizes utility
    imax, umax = argmax(utils)
    imax = np.argmax(utils)
```

Example 4

Example 4: Benchmarking against NumPy's `argmax()`

We want to find out how much slower or faster our own implementation of `argmax()` is compared to NumPy's implementation.

Your turn!

1. Create a Jupyter Notebook `example4.ipynb` or use this template
2. Test the run time of your `argmax()` vs. `np.argmax()` using the following sequences:

   ```
   values1 = [1, 2, 3, 4, 5]
   values2 = np.linspace(0.0, 1.0, 1000)
   values3 = np.random.default_rng(123).random(100000)
   ```

   The array `values3` contains 100,000 random numbers from the interval $[0, 1)$.

   Hint: Use the cell magic `%timeit` to evaluate how long a command executes.
3. Comment on the relative speed of your implementation vs. NumPy's. How does it depend on the data type (list, NumPy array) and the sample size?

Solutions

Solutions to all exercises are available on GitHub:
https://github.com/richardfoltyn/TECH2-H24

Additional Resources

Frequently used terminal commands

These commands work in the macOS Terminal, on Linux, and the Git Bash on Windows:

- `cd` change **d**irectory

```
cd                          # Change to home directory
cd tech2                    # Change to folder "tech2"
cd "Tech 2"                 # Change to folder "Tech 2". Quotes are required if
                            #   the name contains spaces
cd ..                       # Change to the parent directory
```

- `ls` **l**i**s**t files and directories

```
ls                          # List contents of current folder
ls tech2                    # List contents of folder tech2
ls -l example.py            # Show detailed info about file example.py
```

- `pwd` **p**rint **w**orking **d**irectory (i.e., path of the current directory)

Frequently used terminal commands (continued)

- `mkdir` **make dir**ectory

```
mkdir tech2                 # Create folder tech2 in the current folder
mkdir "Tech 2"              # Create folder "Tech 2" in the current folder.
                            #   Quotes are required if the name contains spaces
```

- `rm` **rem**ove file or folder

```
rm example.py               # Delete file example.py in the current folder.
rm -rf tech2                # Delete the folder tech2 and ALL the files and
                            #   sub-folders contained in it.
                            #   WARNING: this cannot be undone.
```

- `cat` print the contents of a file in the terminal

```
cat example1.py             # Print the contents of example1.py
```

Use this command only for text files!

Frequently used git commands — Creating a repository

- `git init` initialize new git repository in current folder

```
git init                   # Initialize git repository, use default name
                           #   for main branch
git init -b main           # Initialize git repository, use "main" as name
                           #   for main branch
```

Frequently used git commands — Adding & removing files

- `git add` add files to git

```
git add example.py         # Add file example.py from the current folder
git add .                  # Add all files in current folder and
                           #   its sub-folders
```

- `git rm` Remove file from git (and your computer)

```
git rm example.py          # Remove example.py from git and delete the file
```

- `git mv` Move (or rename) files

```
git mv example1.py example2.py    # Rename file example1.py to example2.py
```

Frequently used git commands — Creating commits

- `git commit` commit changes that have been added to git

```
git commit                        # Commit changes
                                  #   (opens editor for commit message)
git commit -m "Commit message"    # Commit changes, specify commit
                                  #   message directly
```

Frequently used git commands — Inspecting changes

- `git status` show status of git repository
- `git diff` show differences versus last commit
- `git log` show history of previous commits

Frequently used git commands — Working with remotes

- `git push` push changes to remote repository (e.g., GitHub)

```
git push
git push -u origin main     # Push branch "main" to remote "origin"
```

- `git pull` pull changes from remote repository (e.g., GitHub)
- `git clone` clone a remote repository to your computer

```
git clone https://github.com/richardfoltyn/TECH2-H24.git
```

Video tutorials

Introduction to the command line / terminal:

- Absolute BEGINNER Guide to the **Mac OS** Terminal [17 min]
  https://youtu.be/aKRYQsKR46I
- Git Bash - Simplest command line program for **Windows** [7 min]
  https://youtu.be/yoZ910JQzrg

Introduction to using git

- Git for dummies [20 min] https://youtu.be/mJ-qvsxPHpY
- Git and GitHub Tutorial for Beginners [46 min] https://youtu.be/tRZGeaHPoaw
- Git Essentials in VS Code [30 min] https://youtu.be/twsYxYaQikI
  Focuses on interacting with git and GitHub through VS Code