Mid Journey V4 prompt: a human interacts with a loop-shape object floating in the air, several cubic objects with grid pattern also float in the air around the human --ar 3:2 --niji --q 2

LECTURE 3-1: PYTHON BASICS II - ARRAY

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

2023 APSC-5984 SS: Agriculture Data Science

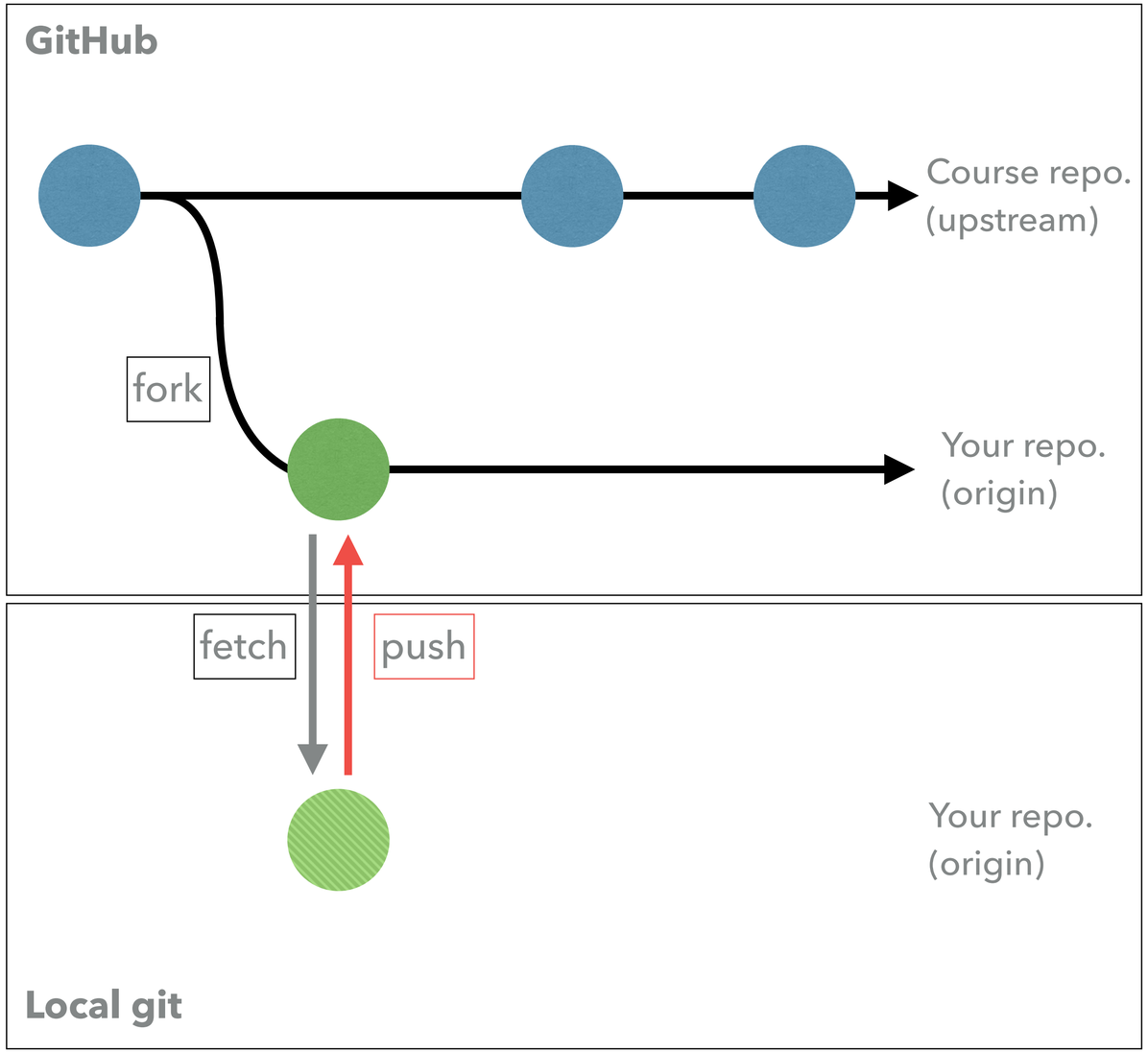
Syncing the Course Repository

First time syncing

Official Guide

Check the current upstream repository:

```
git remote -v
# origin git@github.com:Poissonfish/APSC-5984-ADS.git (fetch)
# origin git@github.com:Poissonfish/APSC-5984-ADS.git (push)
```



Syncing the Course Repository

First time syncing

Official Guide

Check the current upstream repository:

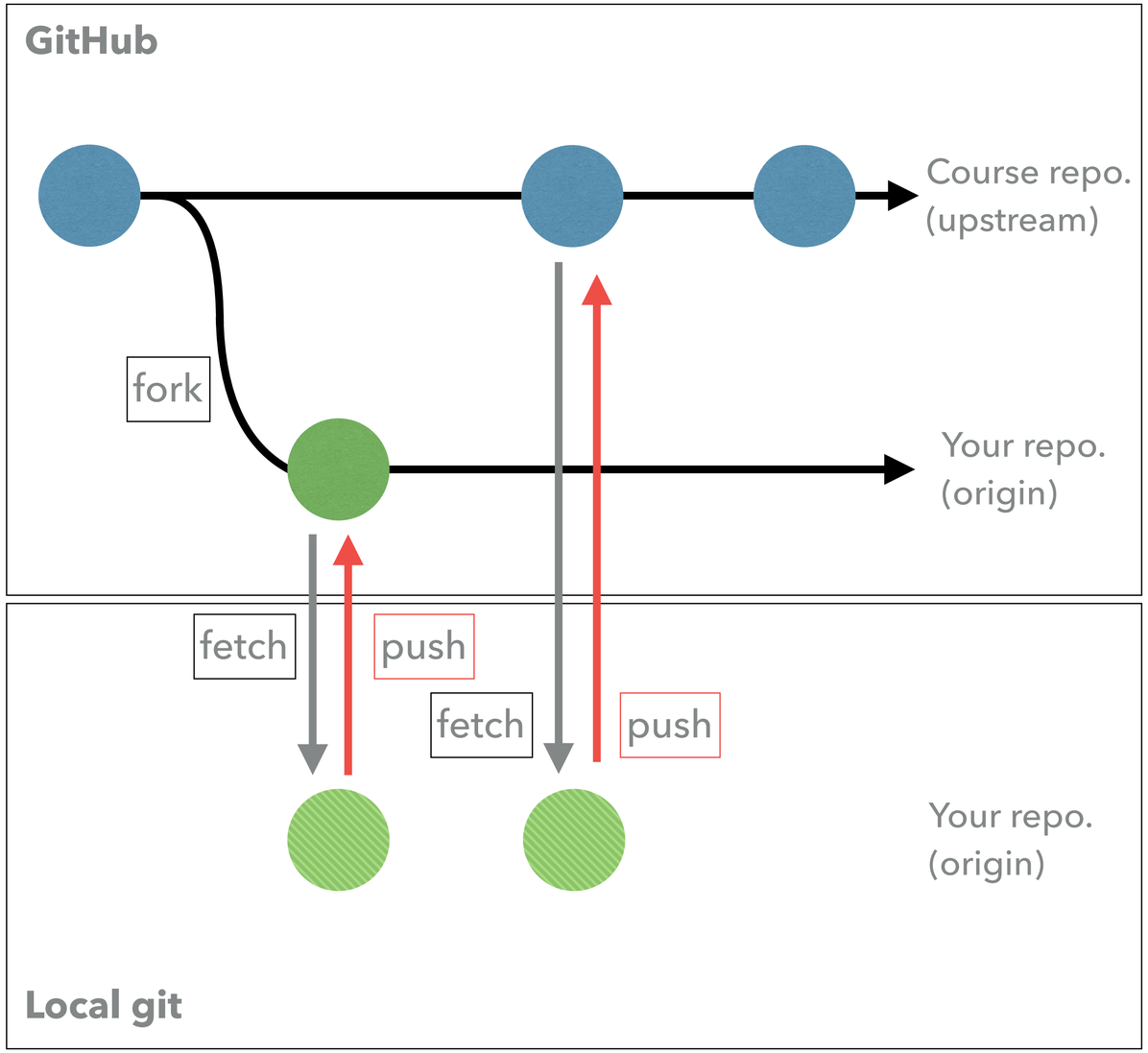
```
git remote -v
# origin git@github.com:Poissonfish/APSC-5984-ADS.git (fetch)
# origin git@github.com:Poissonfish/APSC-5984-ADS.git (push)
```

You won't see the course repository as the upstream repository. So let's add the course repository as the upstream:

```
git remote add upstream git@github.com:Niche-Squad/APSC-5984-ADS.git
```

Check again:

```
git remote -v
# origin git@github.com:Poissonfish/APSC-5984-ADS.git (fetch)
# origin git@github.com:Poissonfish/APSC-5984-ADS.git (push)
# upstream git@github.com:Niche-Squad/APSC-5984-ADS.git (fetch)
# upstream git@github.com:Niche-Squad/APSC-5984-ADS.git (push)
```

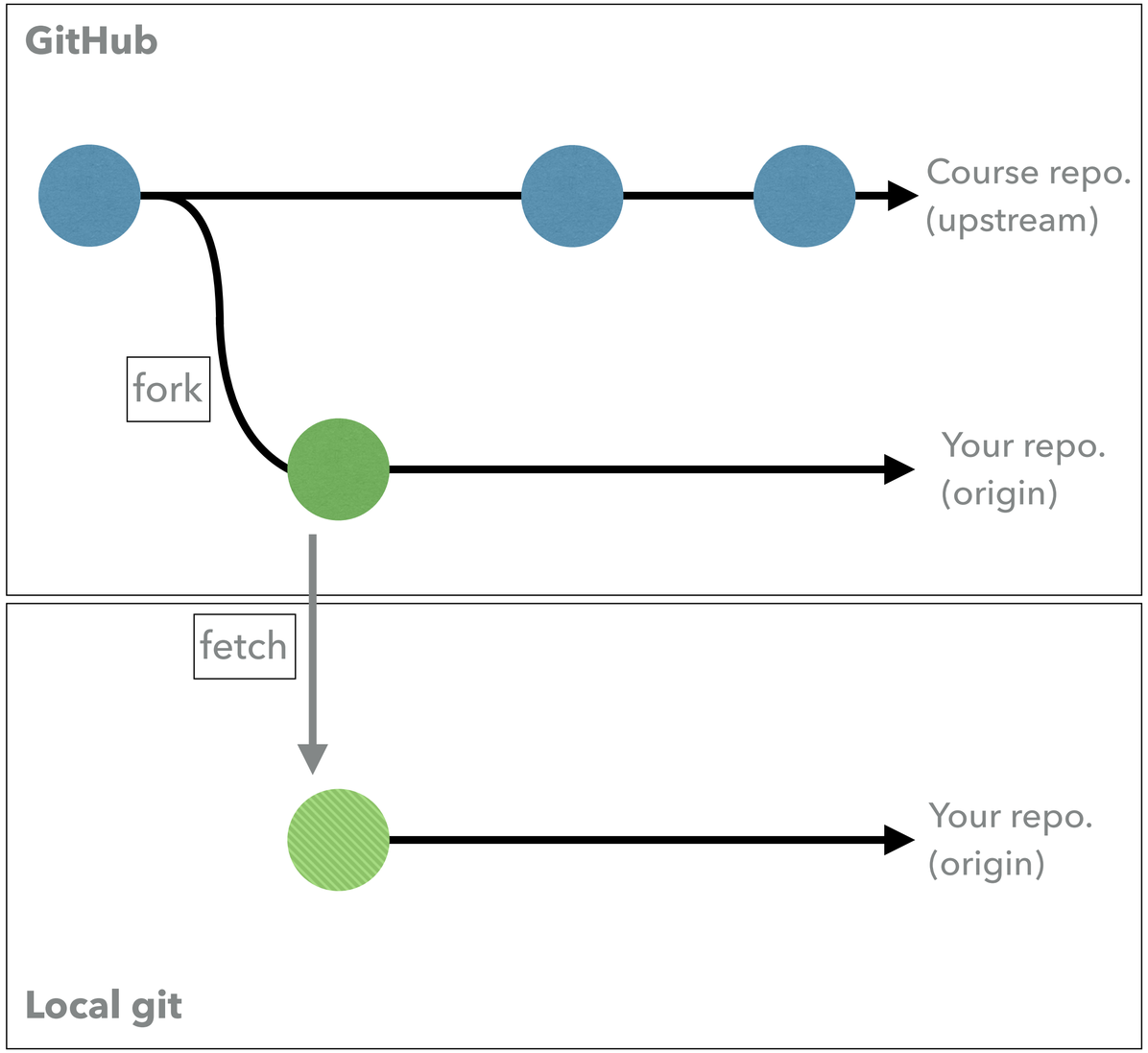


Syncing the Course Repository

Syncing after configuring the upstream repository

Official Guide

```
git fetch upstream
git merge upstream/main --no-edit --allow-unrelated-histories
git push
```

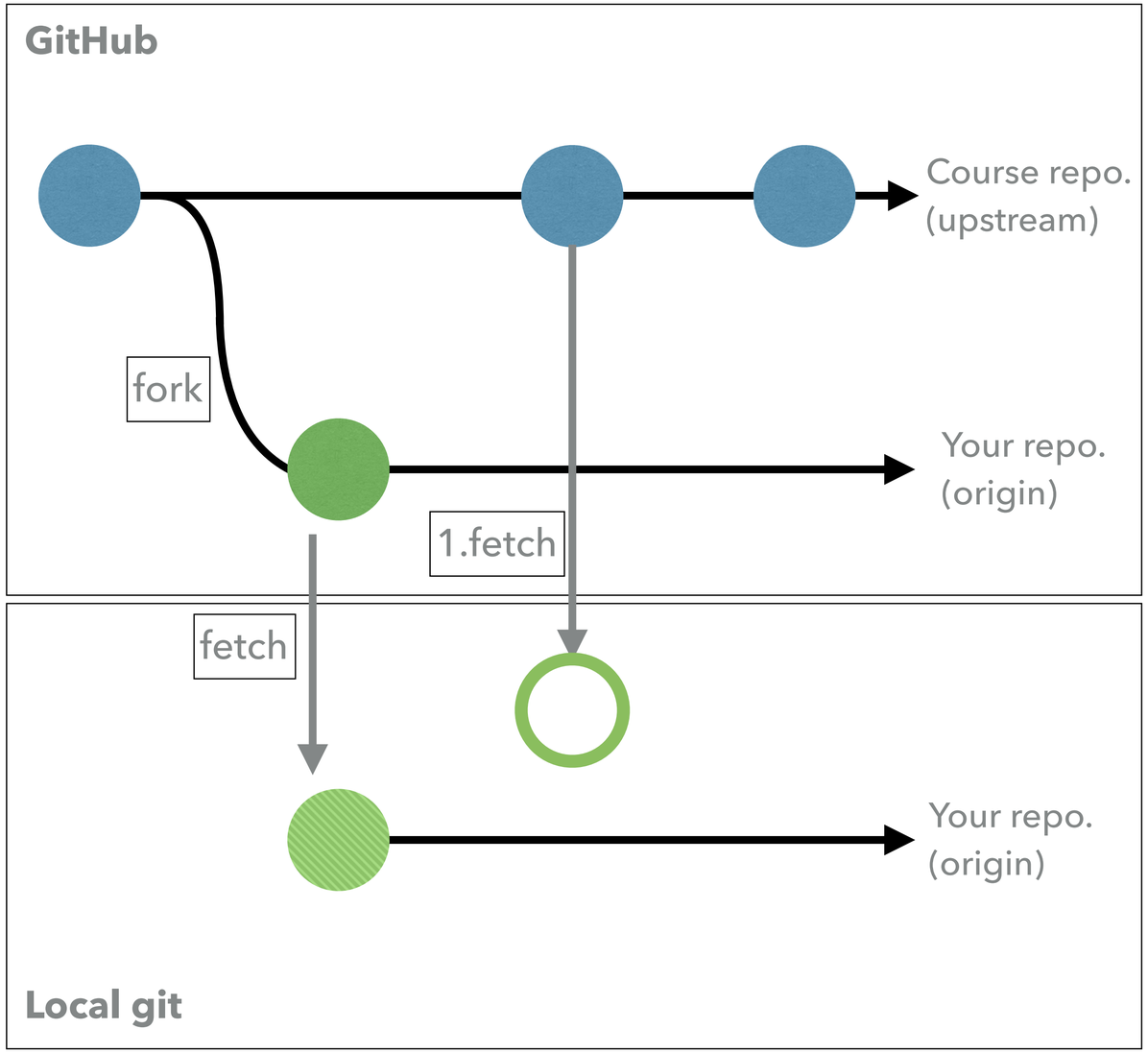


Syncing the Course Repository

Syncing after configuring the upstream repository

Official Guide

```
git fetch upstream
git merge upstream/main --no-edit --allow-unrelated-histories
git push
```

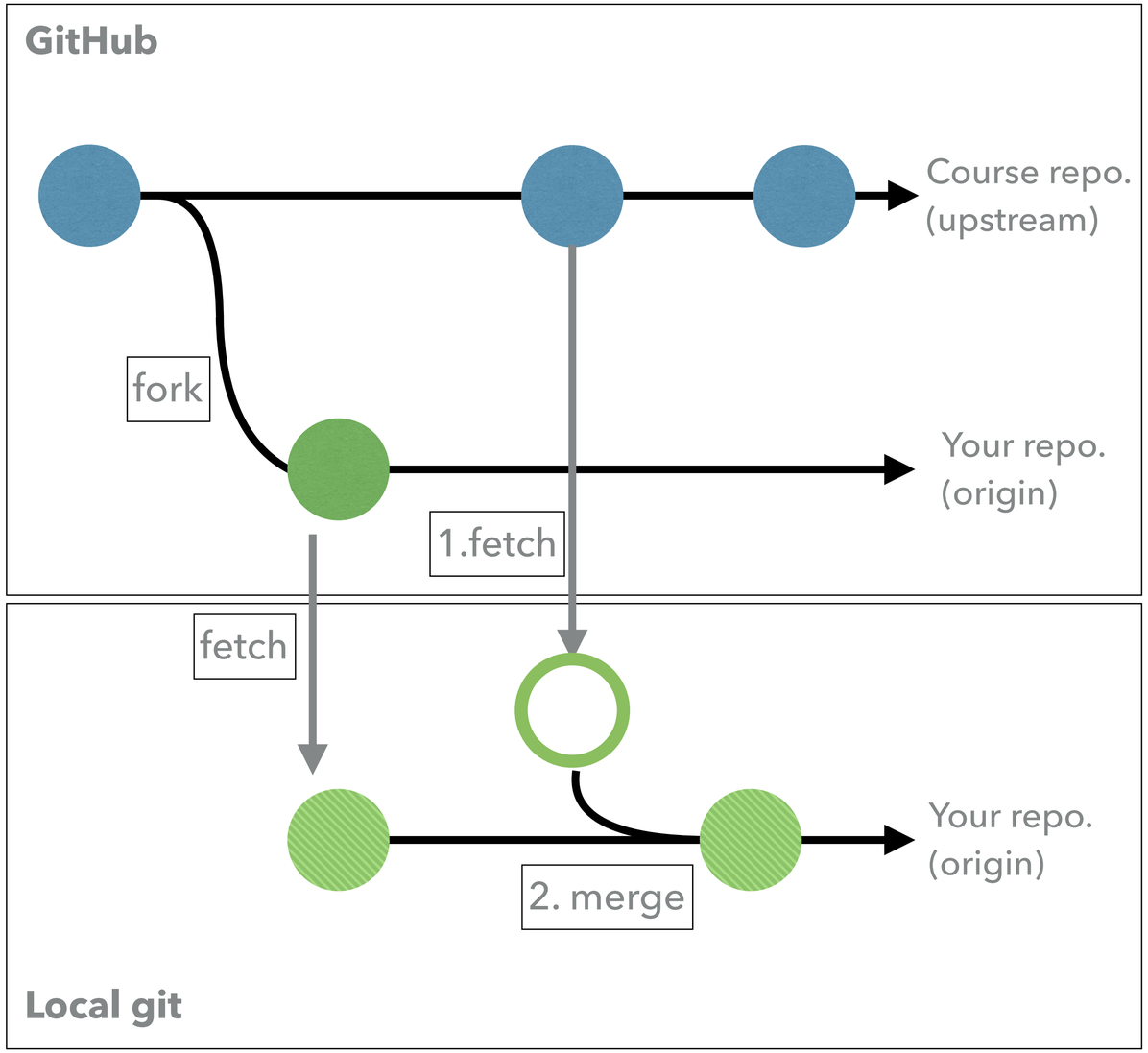


Syncing the Course Repository

Syncing after configuring the upstream repository

Official Guide

```
git fetch upstream
git merge upstream/main --no-edit --allow-unrelated-histories
git push
```

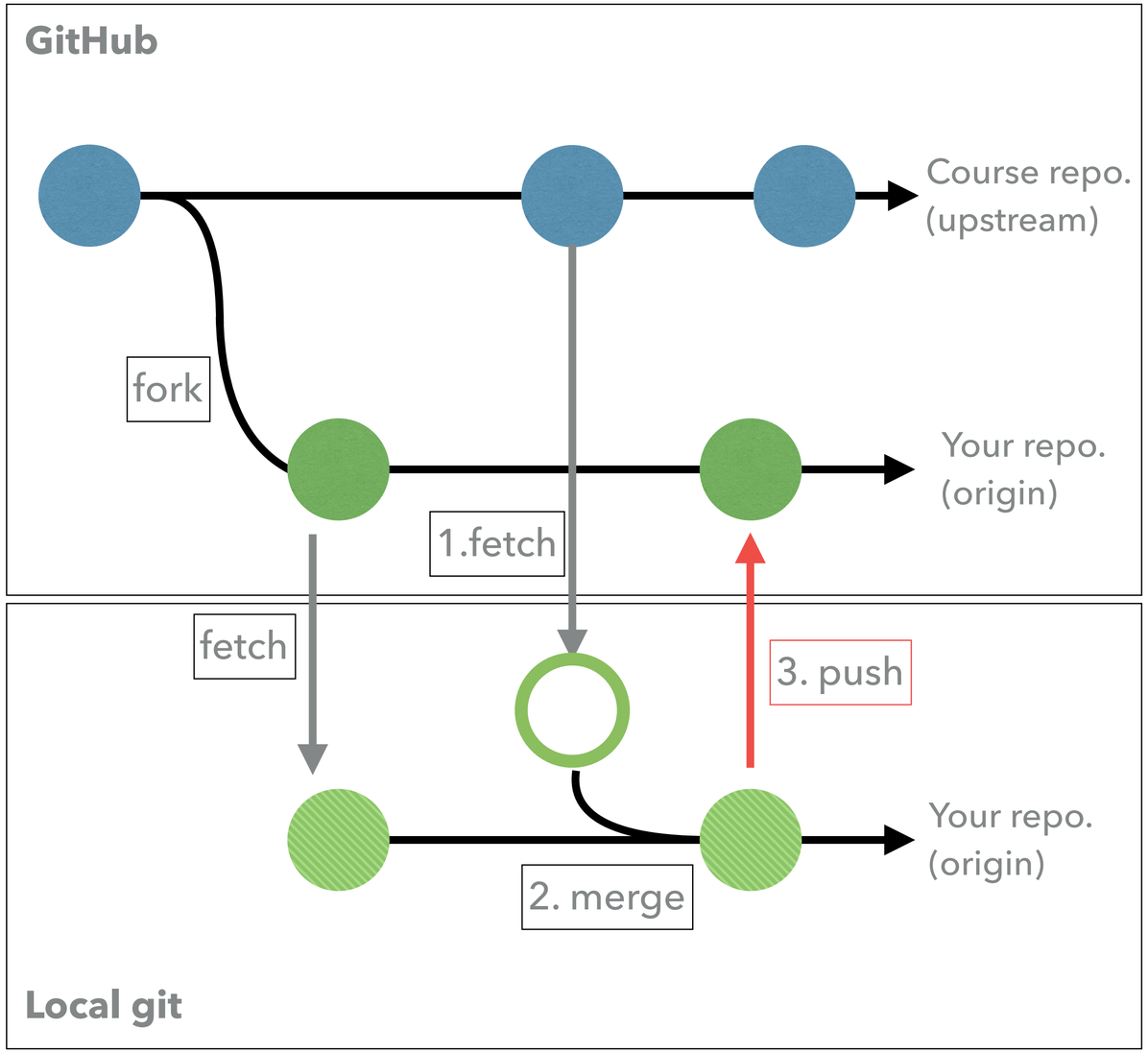


Syncing the Course Repository

Syncing after configuring the upstream repository

Official Guide

```
git fetch upstream
git merge upstream/main --no-edit --allow-unrelated-histories
git push
```



Start-Up Files

e.g., “/home/niche/“ or “/user/niche/

Home directory

~/.bash_profile

This file is executed for login shells.

~/.bashrc

This file is executed for interactive non-login shells.

~/.bash_logout

This file is executed when a login shell exits.

Start-Up Files (Automatically Sync the Repository)

Build your own start-up files

Open Linux terminal, and let's create a file called `.bash_profile` in your home directory (`~`):

```
code ~/.bash_profile
```

This will create a file called `.bash_profile` in your home directory and open it in VS Code.

Now, we can define what to do automatically when you open a terminal. For example, we can add the following lines to the file:

```
# Sync the course repository automatically
cd ~/APSC-5984-ADS # Change the directory to the course repository
git fetch upstream
git merge upstream/main --no-edit --allow-unrelated-histories
git push
cd ~
```

Start-Up Files (Alias)

You can also add aliases to the folders you frequently work on

```
# change WD
export ADS="/home/niche/APSC-5984-ADS" ← Define your path variable
alias toADS="cd $ADS"
# equivalent to `cd /home/niche/APSC-5984-ADS`

# other commands
alias l="ls -lht"
alias ll="ls -l"
alias la="ls -a"
```

Start-Up Files (Example)


```
cd ~/OneDrive\ -\ Virginia\ Tech

# alias
export profile="/Users/niche/.bash_profile"
export arc1="tinkercliffs1.arc.vt.edu"
alias arc="ssh $arc1"

# Github
alias pull="git pull origin master"
push() {
    git add .
    git rm -r --cache *.DS_Store
    git commit -m "${1}"
    git push
}

# inspect data
alias nrow="awk 'END{print NR}'"
alias ncol="awk 'NR==1{print NF}'"
alias shape="awk 'END{print NR, NF}'"
alias peek="awk NR==2,NR==${2}+1{print} ${1}"

# tasks/analysis
alias l="ls -lht"
alias ol="tail *.log"
alias o="tail *.out"
alias t="top -d 1"

# environment
conda activate tf
```

Start-Up Files (Activate)


```
source ~/.bash_profile
```

Overview

- APSC-5984 Lab 3: Python Basics II
  - 0. Overview
  - 1. Lists
    - 1.1 Assign values to a list
    - 1.2 Accessing elements in a list
  - 2. NumPy Array → NumPy
    - 2.1 Create a 1D array
    - 2.2 Multi-dimensional matrix
    - 2.3 Indexing and slicing
    - 2.4 Basic statistics
    - 2.5 Axis-wise operations
  - 3. Dictionaries
    - 3.1 Creating a dictionary
    - 3.2 Accessing elements in a dictionary
  - 4. Loops
    - 4.1 `for` loops
    - 4.2 `while` loops
    - 4.3 `break` and `continue`

List - Instance

strings (characters)

```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters) # ['a', 'b', 'c', 'd', 'e']
```

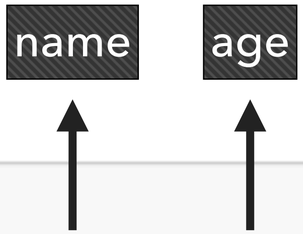
integers

```
numbers = [1, 2, 3, 4, 5]
print(numbers) # [1, 2, 3, 4, 5]
```

mixture

```
mixed = ['a', 1, 'b', 2, 'c', 3]
print(mixed) # ['a', 1, 'b', 2, 'c', 3]
```

List - Nested Structure




```
person_age_1 = ['John', 20]
person_age_2 = ['Mary', 25]
person_age_3 = ['Michael', 30]
nested_list = [person_age_1, person_age_2, person_age_3]
print(nested_list) # [['John', 20], ['Mary', 25], ['Michael', 30]]
```

List - Indexing


```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[0]) # a
```

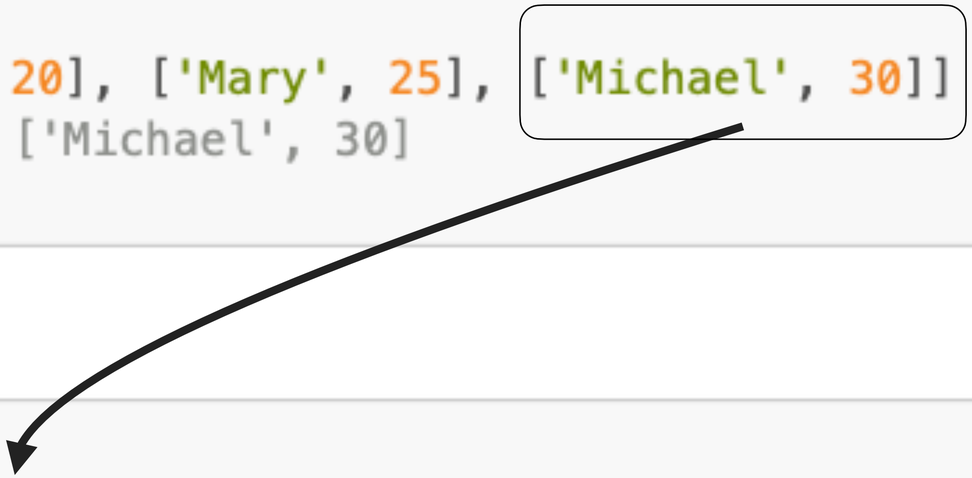
List - Indexing


```
nested_list = [['John', 20], ['Mary', 25], ['Michael', 30]]
print(nested_list[2]) # ['Michael', 30]
```

```
mike_age = nested_list[2][1]
print(mike_age) # 30
```

List - Indexing

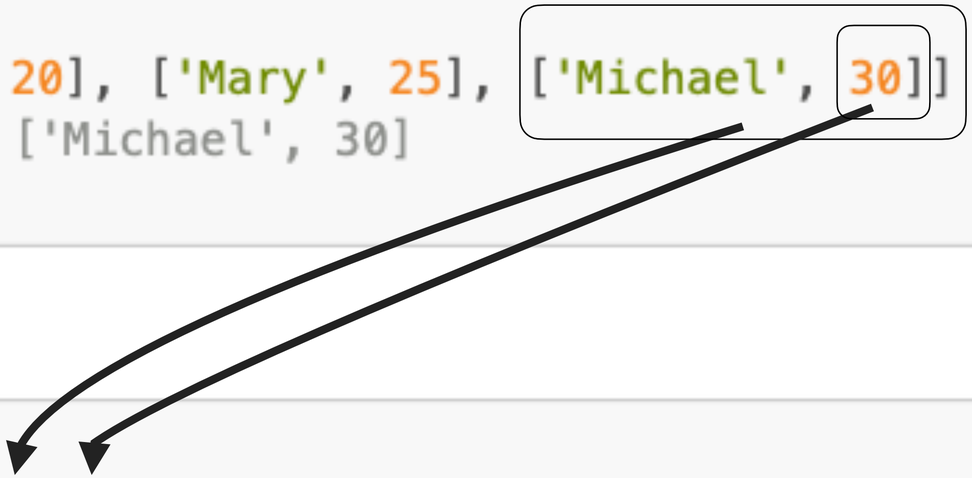

```
nested_list = [['John', 20], ['Mary', 25], ['Michael', 30]]
print(nested_list[2]) # ['Michael', 30]
```



```
mike_age = nested_list[2][1]
print(mike_age) # 30
```

List - Indexing

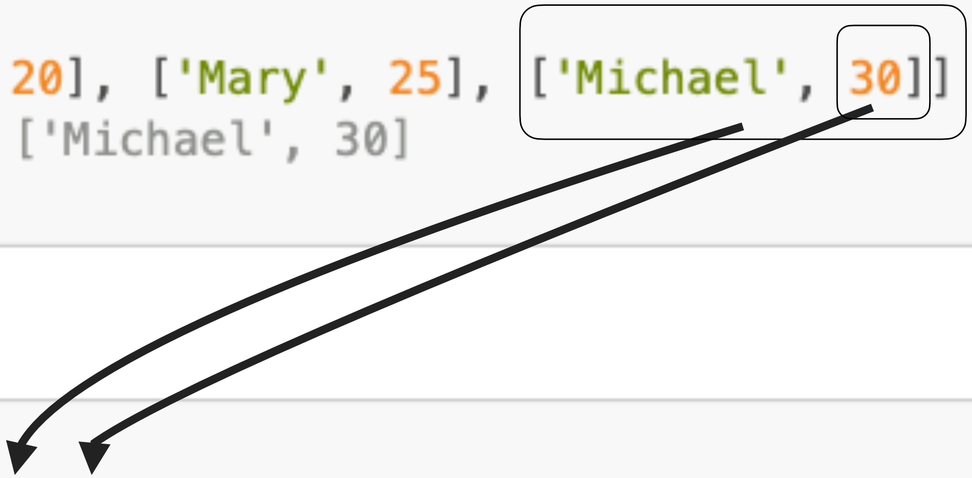

```
nested_list = [['John', 20], ['Mary', 25], ['Michael', 30]]
print(nested_list[2]) # ['Michael', 30]
```



```
mike_age = nested_list[2][1]
print(mike_age) # 30
```

List - Indexing


```
nested_list = [['John', 20], ['Mary', 25], ['Michael', 30]]
print(nested_list[2]) # ['Michael', 30]
```

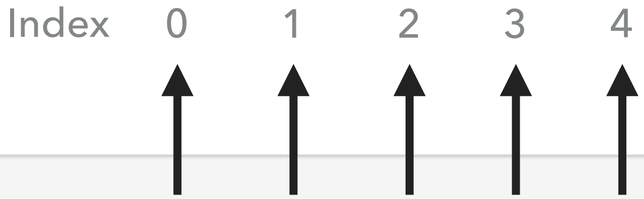


```
mike_age = nested_list[2][1]
print(mike_age) # 30
```

List - Negative Indexing


```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[-1]) # e
```

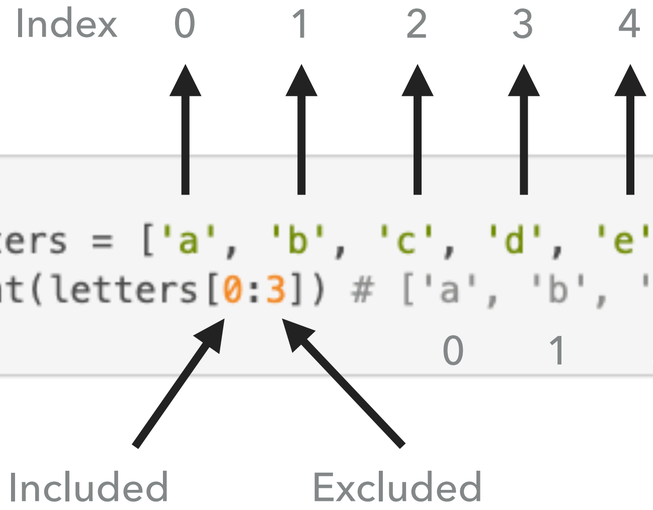
List - Slicing




```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[0:3]) # ['a', 'b', 'c']
```

```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[0:4]) # ['a', 'b', 'c', 'd']
```

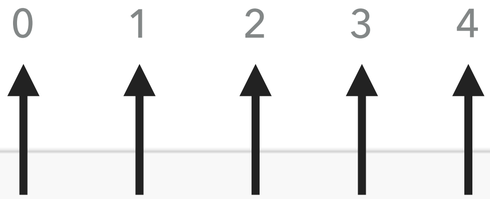
List - Slicing




```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[0:3]) # ['a', 'b', 'c']
```

```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[0:4]) # ['a', 'b', 'c', 'd']
```

List - Slicing




```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[3:]) # ['d', 'e']
```

```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[-3:]) # ['c', 'd', 'e']
```

```
letters = ['a', 'b', 'c', 'd', 'e']
print(letters[1:-1]) # ['b', 'c', 'd']
```

NumPy - Instance


```
import numpy as np
np.__version__
# '1.23.1'
```

2.1 Create a 1D array

Put a list into `np.array()` to create a 1D numpy array.
(https://numpy.org/doc/stable/reference/generated/numpy.array.html)

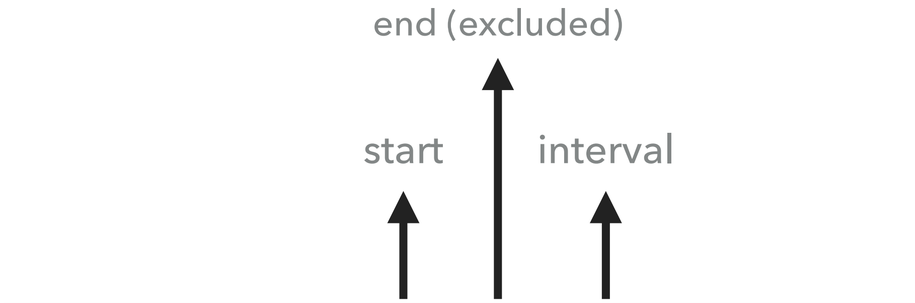
```
ls_1d = [1, 2, 3]
np_1d = np.array(ls_1d)

print(ls_1d)
# [1, 2, 3]
print(np_1d)
# array([1 2 3])
```

NumPy - Instance

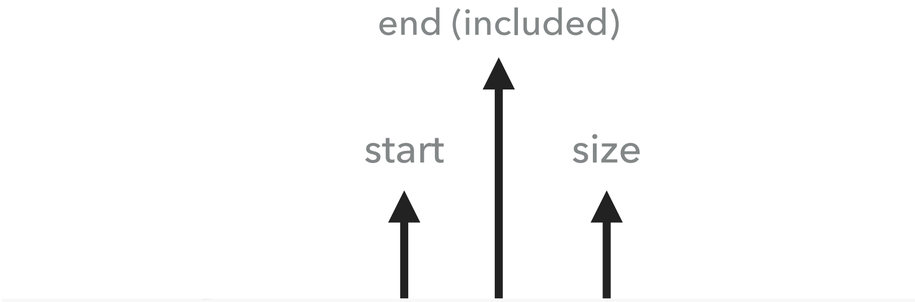
All-zero or all-one array can be created by `np.zeros()` and `np.ones()`.
(https://numpy.org/doc/stable/reference/generated/numpy.zeros.html)
(https://numpy.org/doc/stable/reference/generated/numpy.ones.html)

```
np.zeros(3)
# array([0., 0., 0.])
np.ones(3)
# array([1., 1., 1.])
```

NumPy - Instance (Sequence)




```
np.arange(1, 10, 2)
# array([1, 3, 5, 7, 9])
```




```
np.linspace(1, 10, 5)
# array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

NumPy - Multi-Dimensional Matrix


```
nested_list = [[1, 2, 3], [4, 5, 6]]
np_2d = np.array(nested_list)
print(np_2d)
# array([[1, 2, 3],
#        [4, 5, 6]])
```

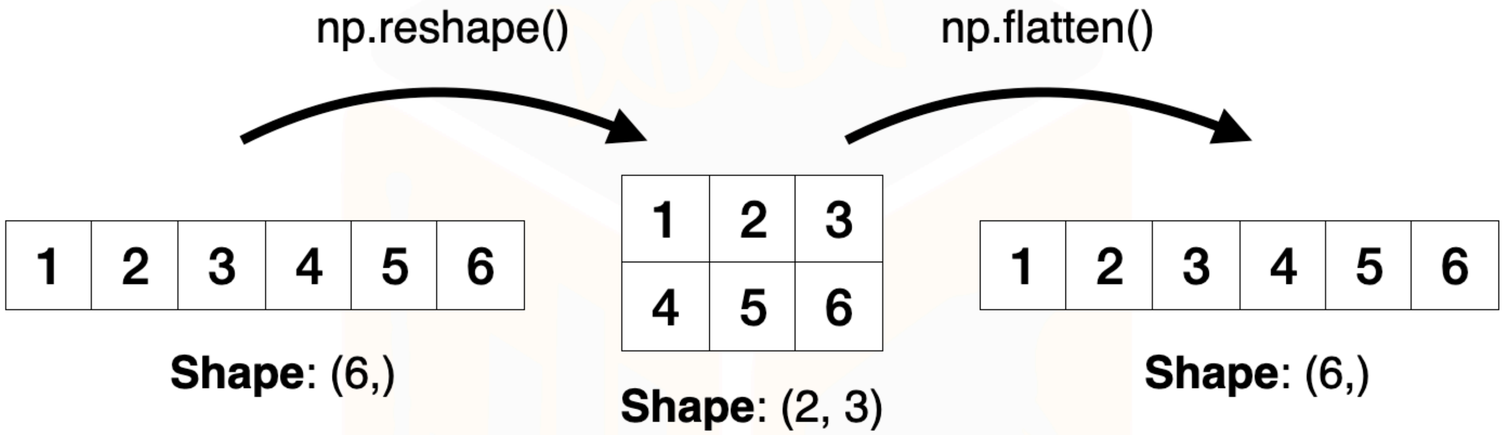
Just like a list

```
np.shape(np_2d)
# (2, 3)
```

NumPy - Multi-Dimensional Matrix

Reshape an array by `np.reshape()` and `np.flatten()`.
(https://numpy.org/doc/stable/reference/generated/numpy.reshape.html)
(https://numpy.org/doc/stable/reference/generated/numpy.flatten.html)

```
np_1d = np.arange(1, 7)
print(np_1d)
# array([1, 2, 3, 4, 5, 6])
np_2d = np_1d.reshape(2, 3)
print(np_2d)
# array([[1, 2, 3],
#        [4, 5, 6]])
print(np_2d.flatten())
# array([1, 2, 3, 4, 5, 6])
```



NumPy - Multi-Dimensional Matrix

Practice with a 3D-array:

```
np_3d = np.arange(1, 13).reshape(2, 2, 3)
print(np_3d)
# array([[[ 1,  2,  3],
#         [ 4,  5,  6]],
#        [[ 7,  8,  9],
#         [10, 11, 12]]])
np_1d = np_3d.flatten()
print(np_1d)
# array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

NumPy - Indexing


```
np_2d = np.linspace(1, 30, 30).reshape((5, 6))
print(np_2d.shape)
# (5, 6)
print(np_2d)
# array([[ 1.,  2.,  3.,  4.,  5.,  6.],
#        [ 7.,  8.,  9., 10., 11., 12.],
#        [13., 14., 15., 16., 17., 18.],
#        [19., 20., 21., 22., 23., 24.],
#        [25., 26., 27., 28., 29., 30.]])
```

```
np_2d[:, 0]
# array([ 1.,  7., 13., 19., 25.])
```

```
np_2d[1:3, :]
# array([[ 7.,  8.,  9., 10., 11., 12.],
#        [13., 14., 15., 16., 17., 18.]])
np_2d[[1, 2], :]
# array([[ 7.,  8.,  9., 10., 11., 12.],
#        [13., 14., 15., 16., 17., 18.]])
```

Double-colon

```
np_2d[:, ::2]
# array([[ 1.,  3.,  5.],
#        [ 7.,  9., 11.],
#        [13., 15., 17.],
#        [19., 21., 23.],
#        [25., 27., 29.]])
```

```
np_2d[-2:, :]
# array([[19., 20., 21., 22., 23., 24.],
#        [25., 26., 27., 28., 29., 30.]])
```

NumPy - Descriptive Statistics


```
np_2d = np.linspace(1, 30, 30).reshape((5, 6))
print(np_2d)
# array([[ 1.,  2.,  3.,  4.,  5.,  6.],
#        [ 7.,  8.,  9., 10., 11., 12.],
#        [13., 14., 15., 16., 17., 18.],
#        [19., 20., 21., 22., 23., 24.],
#        [25., 26., 27., 28., 29., 30.]])
```

Average

Median

STD

```
print(np.mean(np_2d))
# 15.5
print(np.median(np_2d))
# 15.5
print(np.std(np_2d))
# 8.65544144839919
```

Variance

Minimum

Maximum

Sum

```
print(np.var(np_2d))
# 74.91666666666667
print(np.min(np_2d))
# 1.0
print(np.max(np_2d))
# 30.0
print(np.sum(np_2d))
# 465.0
```

NumPy - Axis-Wise Operation


```
np_3d = np.linspace(1, 30, 30).reshape((3, 2, 5))
print(np_3d)
# array([[[ 1.,  2.,  3.,  4.,  5.],
#         [ 6.,  7.,  8.,  9., 10.]],
#        [[11., 12., 13., 14., 15.],
#         [16., 17., 18., 19., 20.]],
#        [[21., 22., 23., 24., 25.],
#         [26., 27., 28., 29., 30.]]])
print(np.mean(np_3d, axis=0))
# array([[11., 12., 13., 14., 15.],
#        [16., 17., 18., 19., 20.]])
print(np.mean(np_3d, axis=1))
# array([[ 3.5,  4.5,  5.5,  6.5,  7.5],
#        [13.5, 14.5, 15.5, 16.5, 17.5],
#        [23.5, 24.5, 25.5, 26.5, 27.5]])
print(np.mean(np_3d, axis=2))
# array([[ 3.,  8.],
#        [13., 18.],
#        [23., 28.]])
```

```
print(np.mean(np_3d, axis=(0, 2)))
# array([12., 17., 22., 27.])
```