Hello, Python!

A quick introduction to
Python syntax, variable assignment, and numbers

LECTURE: 02

[Slides from Fardina Fathmiul Alam]

Importance

- **Python**: It's a user-friendly language for data analysis.
- **Git**: Helps manage code and data changes when working with a team.
- **Pandas**: Simplifies data cleaning and manipulation.
- **Databases**: Needed for storing and retrieving data efficiently.

These skills are essential for effective data analysis, collaboration, and handling data in real-world projects.

Topics to Cover

Introduction to Python

- Syntax
- Variable
- Assignment
- Numbers
- In -class exercises (LATER TOPIC)

Before we get started

We're going to be working in Google Colab today and also for the course!

Quick Note: Google Colab is a **free, cloud-based platform** provided by Google that allows you to write and execute Python code in a web-based interactive environment. It is particularly popular in the field of machine learning and data analysis;

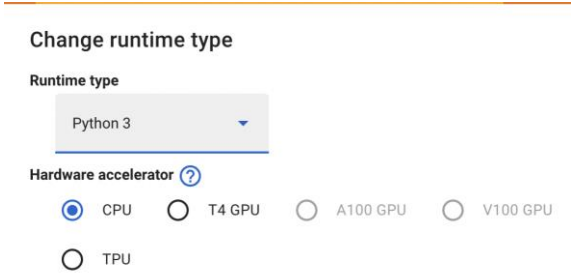
- requires no setup to use
- provides free access to computing resources, including GPUs and TPUs.
- allows multiple people to work on the same notebook simultaneously

Before we get started

We're going to be working in Google Colab today and also for the course!

To load yours, do the following:

- **Sign in** with your Google account
- Once signed in, click on the **"New Notebook"** button. This will create a new Jupyter notebook hosted on Google Colab. **Rename** it as "**DATA602-class1-Your_UID.ipynb**"
- **Default Runtime** is given: Python3 (with CPU).
- If you **want to change it**; Go to the "Runtime" menu at the top and click on "Change runtime type."
- Use Colab's notebook cells to **write and execute** Python code.
- Use "+code/+text" to **create new** code block/ comment block
- **Execute a cell** by clicking 'Play' button next to the cell or using 'Shift + Enter'.
- **Save** manually through 'File' or let Colab auto-save to Google Drive for work preservation.



More Details:
https://colab.research.google.com/drive/16pBJQePbqkz3QFV54L4NIkOn1kwpuRrj

Before we get started

There is another populer alternative: Jupyter Notebook !

Quick Note: Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

- It runs on your local machine, and you can access it through your web browser.
  - Means you have full control over your environment and can easily install any libraries or packages you need.

Check:https://medium.com/@jknrjayalath/jupyter-notebook-vs-google-colab-colaboratory-77dbbafdc636

Before we get started

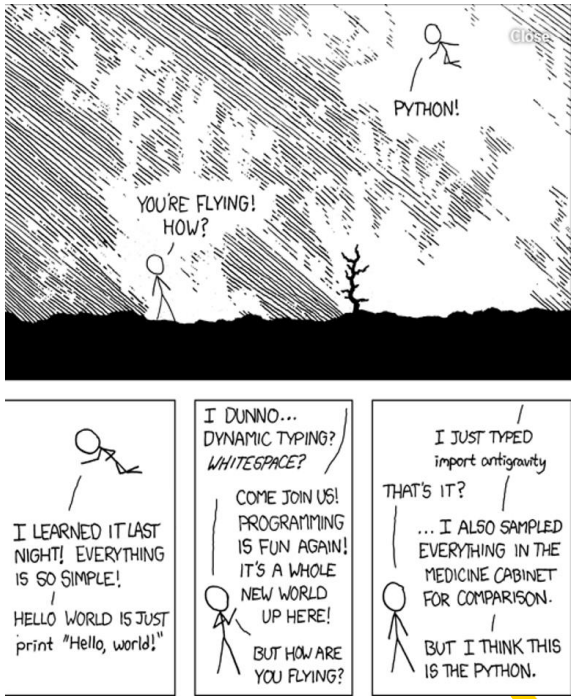
There is another populer alternative: Jupyter Notebook !

- Ensure Python is installed on your system. Download from python.org.
- Open a command prompt or terminal.
- **Install:** Run the following command to install Jupyter: **pip install jupyter**
- **Start:** In the command prompt or terminal, type jupyter notebook and press Enter. A new browser window/tab will open with the Jupyter Notebook interface.
- **Using Jupyter Notebook:** Create a new notebook, add cells for code or text, and execute code interactively.
- Use Markdown cells for narrative text and visualizations.
- Save and share notebooks in various formats, including HTML and PDF.

What is Python?

Python is a Interpreted (i.e. non-compiled), high-level programming language

- A widely used programming language known for its readability and simplicity.
- Interpreter **executes source code directly**; do not require a separate compilation step
- Designed to be **intuitive** and **easy to program in** (read/write)
- **Supports dynamic typing**, allowing variables to change types during runtime
- Also has a **focus on importing modules**, a feature that makes it useful for scientific computing.
- **Runs** on **most operating system**s and **platforms**
- Slow to run



Getting Started with Python

- Download and **install Python 3** from python.org.
  - Note: Python 2.7 is no longer maintained; transition to Python 3.
- Install useful Dependencies: Use pip to install essential data science libraries
  - **pip install** numpy, matplotlib, scipy, pandas, sklearn, …
- **Download an IDE** of your choice
  - Visual Studio, PyCharm, Sublime Text, Spyder, etc.
  - https://stackoverflow.com/questions/81584/what-ide-to-use-for-python
- Run interactively in a Jupyter notebook, Google Colab

Recap: For this course, we will mostly use Google Colab.

A Python Code

```
# This is a comment

def add(a, b):
    return a + b

def main():

    # Count to ten
    i = 0
    while i < 10:
        print(i)

    # Count to ten a different way
    for i in range(0, 10):
        print(i)

    # Printing a list
    myList = [1, 2, 3, 4]
    for some_item in my_list:
        print(some_item)

    return
```

This is unintentionally an infinite loop, needs an i += 1

Try! What's the output?

```
spam_amount = 0
print(spam_amount)

# Ordering Spam, egg, Spam, Spam, bacon and Spam (4 more servings of Spam)
spam_amount = spam_amount + 4

if spam_amount > 0:
    print("But I don't want ANY spam!")

viking_song = "Spam " * spam_amount
print(viking_song)
```


Variables and Types

Dynamically typed: the type of the variable is derived from the value it is assigned.

- Numbers
  - Integer
  - Float
  - Complex
- Boolean
- String

A variable is assigned using the = operator; i.e:

```
intVar = 5
floatVar = 3.2
stringVar = "Food"

print(intVar)
print(floatVar)
print(stringVar)
```

```
In [4]: runfi
lb690/.spyder
5
3.2
Food
```

- The print() function is used to print something to the screen.

Basic Operators

Arithmetic Operators

- + #addition
- - #subtraction
- * #multiplication
- ** # power
- % # modulus
- “in” # check if element is in container

Increment Operator: x += y

Others: x *= y, x -= y etc.

Boolean Operators (useful for conditional statements)

- and, or,not

Comparison Operators

- Greater than: >
- Lesser than: <
- Greater than or equal to: >=
- Lesser than or equal to: <=
- Is equal to: ==

Examples: Basic Operators

Integers

In [1]:
```
1 + 1
```
Out[1]: 2

In [2]:
```
a = 4
```

Floats

In [4]:
```
1.4 + 2.2
```
Out[4]: 3.6

In [5]:
```
c = -2.4 + 1
print(c)
```
```
-1.4
```

Complex

In [8]:
```
a = 1.5 + 0.6j
print(a.real)
print(a.imag)
```
```
1.5
0.6
```

Booleans

```
2.5 < 10
```
True

```
test = (3 > 4)
print(test)
```
False

```
type(test)
```
bool

Strings

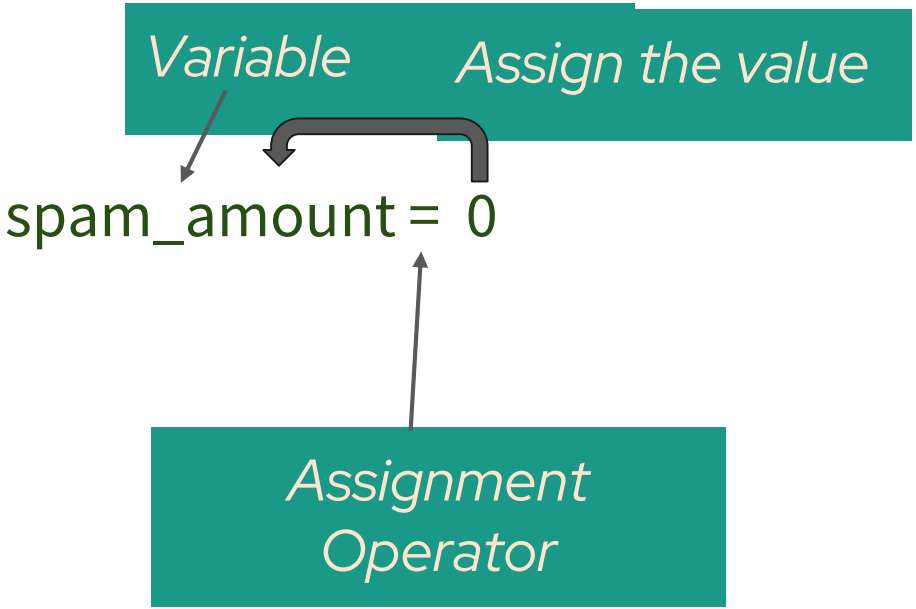
```
my_str = 'Hello World!'
print(my_str)
print(my_str[0])
```
```
Hello World!
H
```

```
print(my_str.replace('World', 'Why\'N\'How'))
print(my_str)
my_str = my_str.replace('World', 'Why\'N\'How')
print(my_str)
```
```
Hello Why'N'How!
Hello World!
Hello Why'N'How!
```

In:
```
intVar = 5
floatVar = 3.2
stringVar = "Food"

if intVar > floatVar:
    print("Yes")

if intVar == 5:
    print("A match!")
```

Out:
```
In [9]: run
lb690/.spyd
Yes
A match!
```


Variable Assignment



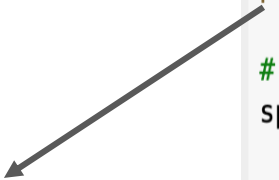
Interesting Facts:

No need to

- ➔ explicitly "declare" a variable before assigning a value to it.
- ➔ "specify" the type of value the variable will refer to; can re-assign it to different types (string, boolean etc.)

Function Call

print(spam_amount)



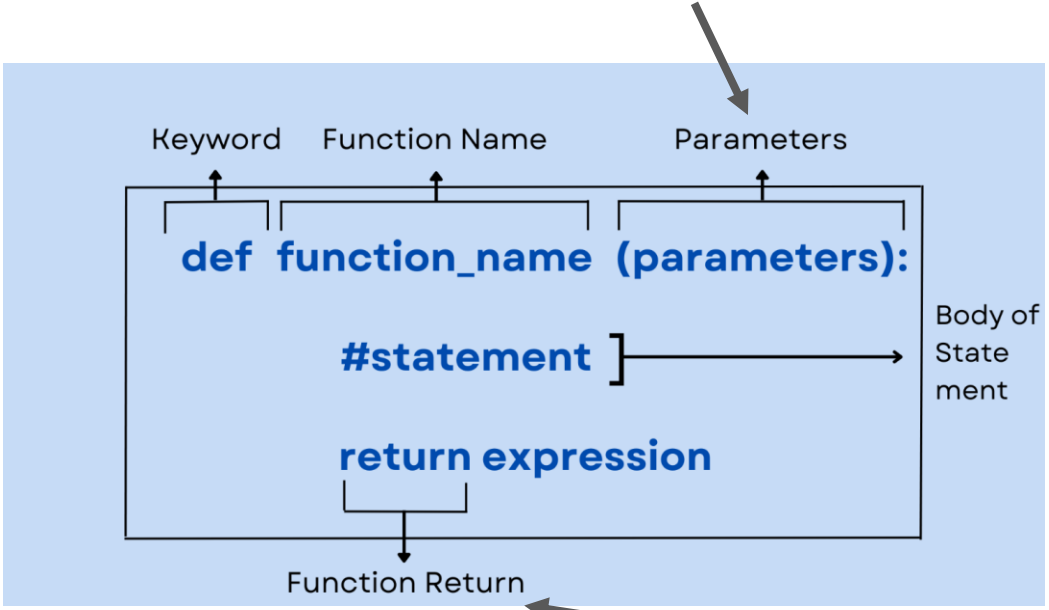
```
spam_amount = 0
print(spam_amount)

# Ordering Spam, egg, Spam, Spam, bacon and Spam (4 more servings of Spam)
spam_amount = spam_amount + 4

if spam_amount > 0:
    print("But I don't want ANY spam!")

viking_song = "Spam " * spam_amount
print(viking_song)
```

What is Function in Python?

A function is a block of code which only runs when it is called.

- Two types of functions:
  - **User-defined functions –** The user defines these functions to carry out a certain task.
  - **Built-in functions –** These Python functions are built-in functions.

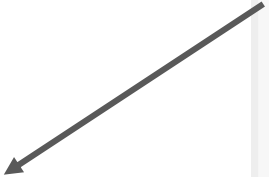
Function Structure



A function can return data as a result

Built-in functions: function Call

print(spam_amount)



```
spam_amount = 0
print(spam_amount)

# Ordering Spam, egg, Spam, Spam, bacon and Spam (4 more servings of Spam)
spam_amount = spam_amount + 4

if spam_amount > 0:
    print("But I don't want ANY spam!")

viking_song = "Spam " * spam_amount
print(viking_song)
```

User-defined functions

```
# User-defined function with .split() method to count words

def count_words(sentence):
    words = sentence.split()
    return len(words)

# Example usage
input_sentence = "Python is a versatile programming language."
word_count = count_words(input_sentence)

# Print the result
print("Number of words:", word_count)
```

- Functions (Custom) operations that take one or more pieces of data as arguments
- Methods: functions called directly off data using the “.” operator (ex: .split())

Function Call cont.

The function must be defined before it is called.

- In other words, the block of code that makes up the function must come before the block of code that makes use of the function.

In:

```
def practice_function(a, b):
    answer = a * b
    return answer

x = 5
y = 4
calculated = practice_function(x, y)
print(calculated)
```

Out:

```
In [10]: runfi
lb690/.spyder-
20
```

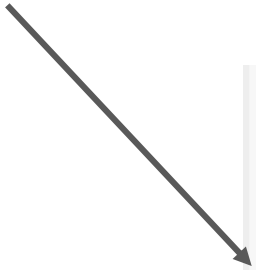
In:

```
def multiply_function(a, b):
    result = a * b
    return result

numbers_list = [1, 2, 3]
multiplier_list = [2, 4]
for n in numbers_list:
    print("___________")
    for m in multiplier_list:
        current_answer = multiply_function(n, m)
        print("The answer to %d * %d is: " %(n, m), current_answer)
```

```
___________
The answer to 1 * 2 is:  2
The answer to 1 * 4 is:  4
___________
The answer to 2 * 2 is:  4
The answer to 2 * 4 is:  8
___________
The answer to 3 * 2 is:  6
The answer to 3 * 4 is:  12
```

Comment

Begin with the # symbol.



Reassign the value of an existing variable

```
spam_amount = 0
print(spam_amount)

# Ordering Spam, egg, Spam, Spam, bacon and Spam (4 more servings of Spam)
spam_amount = spam_amount + 4

if spam_amount > 0:
    print("But I don't want ANY spam!")

viking_song = "Spam " * spam_amount
print(viking_song)
```

Python Data Structures

- Lists (ordered, mutable set of objects): var = ['one', 1, 1.0]
- Tuples (ordered, immutable set of objects): var = ('one', 1, 1.0)
- Dictionaries (Unordered lists of key-valued pairs): var = {'one': 1, 'two': 2, 1: 'one', 2: 'two'}

Each of the above has its own set of methods. E.g: Dictionaries methods are keys(), values(), items(), etc.

Examples: Python Data Structures

```
numbers = [1, 2, 3]
print("List 1:", numbers)
numbers[1] = 5
print("Amended list 1:", numbers)
```

```
List 1: [1, 2, 3]
Amended list 1: [1, 5, 3]
```

In:

```
prices = {"Eggs": 2.30,
          "Steak": 13.50,
          "Bacon": 2.30,
          "Beer": 14.95}
print("1:", prices)
print("2:", type(prices))
print("The price of bacon is:", prices["Bacon"])
```

Out:

```
1: {'Eggs': 2.3, 'Steak': 13.5, 'Bacon': 2.3, 'Beer': 14.95}
2: <class 'dict'>
The price of bacon is: 2.3
```

```
tuple1 = (5, 10)
print('1:', tuple1)
print("2:", type(tuple1))
```

```
1: (5, 10)
2: <class 'tuple'>
```

```
tuple1[1] = 6
```

```
TypeError: 'tuple' object does not support item assignment
```


Adding elements to a list

- **Use of an index** to replace a current element with a new one.

```
In:  numbers = [1, 2, 3]
     print("List 1:", numbers)
     numbers[1] = 5
     print("Amended list 1:", numbers)

Out: List 1: [1, 2, 3]
     Amended list 1: [1, 5, 3]
```

- Use the insert() function in order to add an element to a list at a specific indexed location, without overwriting any of the original elements.

```
In:  numbers = [1, 2, 3]
     print("List 1:", numbers)
     numbers.insert(2, 'Surprise!')
     print("Amended list 1:", numbers)

Out: List 1: [1, 2, 3]
     Amended list 1: [1, 2, 'Surprise!', 3]
```

- Add an element to the end of a list using the append() function

```
In:  numbers = [1, 2, 3]
     print("List 1:", numbers)
     numbers.append(4)
     print("Amended list 1:", numbers)

Out: List 1: [1, 2, 3]
     Amended list 1: [1, 2, 3, 4
```

Try: Replace the second element in your string with the integer 2.
Try: Use insert() to put the integer 3 after the 2 that you just added to your string.
Try: Use append() to add the string “end” as the last element in your list.

Removing elements from a list

You can remove an element using remove() from a list based upon the element value.

- Remember: If there is more than one element with this value, **only the first occurrence** will be removed.

In:
```
numbers = [1, 2, 3, 3]
print("List 1:", numbers)
numbers.remove(3)
print("Amended list 1:", numbers)
```

Out:
```
List 1: [1, 2, 3, 3]
Amended list 1: [1, 2, 3]
```

Also remove elements by their index using the **del function.**

In:
```
numbers = [1, 2, 3, 4]
print("List 1:", numbers)
del numbers[1]
print("Amended list 1:", numbers)
del numbers[-1]
print("Amended list 2:", numbers)
```

Out:
```
List 1: [1, 2, 3, 4]
Amended list 1: [1, 3, 4]
Amended list 2: [1, 3]
```

Try: Use del to remove the 3 that you added to the list earlier.

Indexing

- Indexing in Python is 0-based, meaning that the first element in a string, list, array, etc, has an index of 0. The second element then has an index of 1, and so on.

```
In:  test_string = "Dogs are better than cats"
     print('First element:', test_string[0])
     print('Second element:', test_string[1])
```

```
Out: First element: D
     Second element: o
```

- You can cycle backwards through a list, string, array, etc, by placing a minus symbol in front of the index location.

```
In:  test_string = "Dogs are better than cats"
     print('Last element:', test_string[-1])
     print('Second to last element:', test_string[-2])
```

```
Out: Last element: s
     Second to last element: t
```


Slicing

In Python, slicing is a way to **extract a portion of a sequence** (like a list, tuple, or string).

- The syntax: **start:stop:step.** Step is optional; If step is not specified, it defaults to 1

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using slicing to get elements from index 4 to the end
subset_list = my_list[4:]

# Print the result
print(subset_list)

# Using slicing to get elements from the beginning up to (, but not including) index 4
subset_list = my_list[:4]

# Print the result
print(subset_list)
```

Output: [5, 6, 7, 8, 9, 10]

Output: [1, 2, 3, 4]
Up to (but not including) the element at index 4

Try: Slicing

- Create a string that is 10 characters in length.
- Print the second character to the screen.
- Print the third to last character to the screen.
- Print all characters after the fourth character.
- Print characters 2-8.

Python For and While Loops

The for loop is used to **iterate over elements in a sequence**, and is often used when you have a piece of code that you want to repeat a number of times.

For loops essentially say:
"For all elements in a sequence, do something"

```
species = ['dog', 'cat', 'shark', 'falcon', 'deer', 'tyrannosaurus rex']
for i in species:
    print(i)
```

```
numbers = [1, 20, 18, 5, 15, 160]
total = 0
for value in numbers:
    total = total + value
print(total)
```

The while loop tells the computer to do something as long as a specific condition is met.

It essentially says:
"while this is true, do this."

```
species = ['dog', 'cat', 'shark', 'falcon', 'deer', 'tyrannosaurus rex']
i = 0
while i < 3:
    print(species[i])
    i = i + 1
```

```
In [11]: runfile('//is
dog
cat
shark
```

Functions: range()

The range() function generates a list of numbers, which is generally **used to iterate over within for loops**.

Syntax: range(stop)

stop: Number of integers (whole numbers) to generate, starting from 0.

```
for i in range(5):
    print(i)

In [6]: runfile('
0
1
2
3
4
```

Syntax: range([start], [stop], step)

- start: Starting number of the sequence.
- stop: Generate numbers up to, but not including this number.
- step: Difference between each number in the sequence

```
for i in range(3,6):
    print(i)

In [7]: runfile('/
3
4
5
```

```
for i in range(4, 10, 2):
    print(i)

In [8]: runfile
4
6
8
```

Note:

- All parameters **must be integers** (can be positive or negative).
- The range() function (and Python in general) is **0-index** based (list indexes start at 0, not 1).
- **Eg. mylist[0] = the first element of a list.** Therefore the **last integer** generated by range() is up to, but **not including, stop.**

Functions: break. continue

the break() function: terminate a loop sd it breaks out of the innermost enclosing for or while loop.

```
for i in range(1, 10):
    if i == 3:
        break
    print(i)
```

```
In [9]: runfile('//isa
1
2
```

The continue() statement is used to tell Python to **skip the rest of the statements in the current loop block,** and then to **continue** to the **next iteration** of the loop

```
for i in range(1, 10):
    if i == 3:
        continue
    print(i)
```

```
In [10]: runfile('//isa
1
2
4
5
6
7
8
9
```


Nested loops

In some situations, you may want a loop within a loop; this is known as a nested loop.

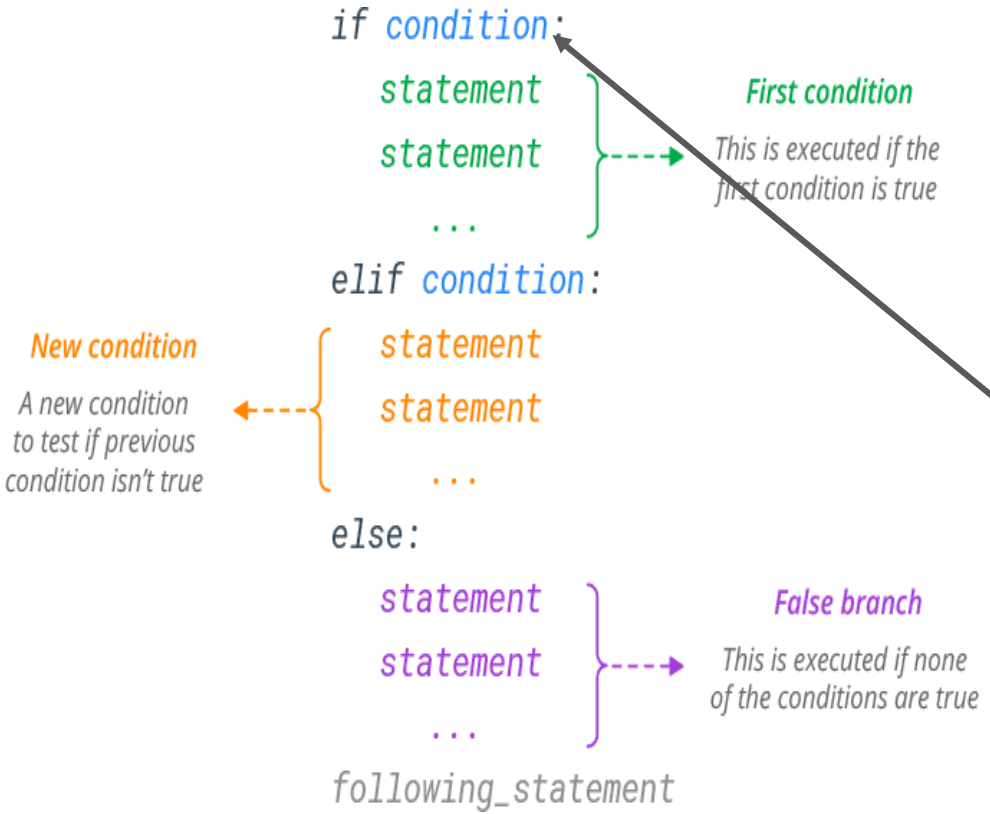
In:

```
for x in range(1, 11):
    for y in range(1, 11):
        print('%d * %d = %d' %(x, y, x*y))
```

Out:

```
In [16]: runfile('//i
 1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
```

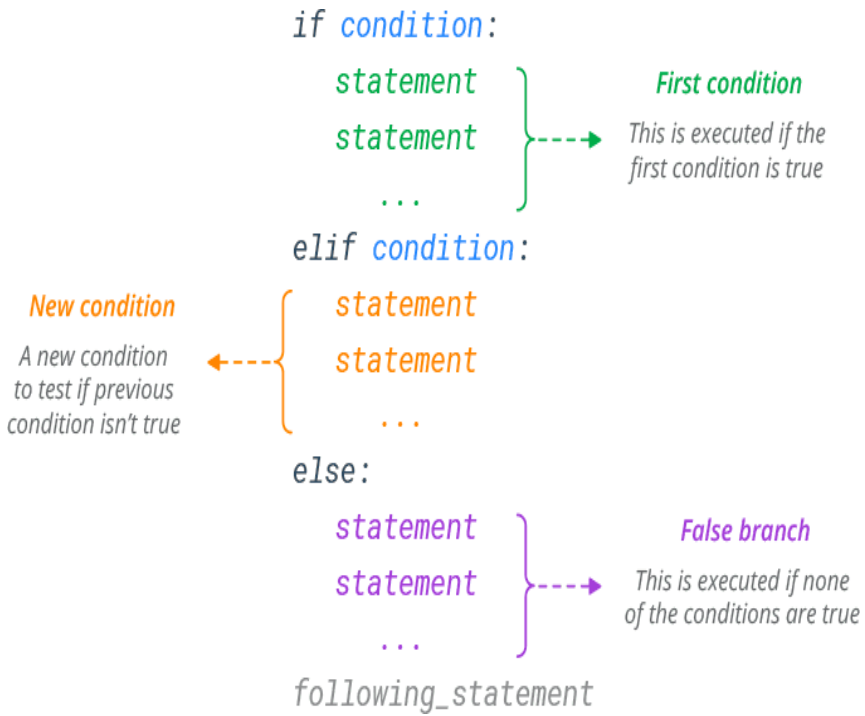
Python Condition: If-Elif-Else Statement



There are three main conditional statements in Python; if, else, elif.

```
spam_amount = 0
print(spam_amount)

# Ordering Spam, egg, Spam, Spam, bacon and Spam (4 more servings of Spam)
spam_amount = spam_amount + 4

if spam_amount > 0:
    print("But I don't want ANY spam!")

viking_song = "Spam " * spam_amount
print(viking_song)
```


If-Elif-Else Statement



Imagine, you need to write a python program t decipher messages from friendly aliens. Yo want to decode an alien's mood based on th number of eyes and tentacles it has.

- ❑ **If the alien has 3 eyes and up to tentacles, it's curious and friendly.**
- ❑ **If the alien has more than 3 eyes an more than 5 tentacles, it might b grumpy.**
- ❑ **But if the alien has 2 or fewer eyes or n tentacles, it seems shy or possibl invisible.**
- ❑ **For other cases, the mood is uncertain and it's an alien mystery.**

Task: Write down the code using if-elif-els conditions to accomplish this alien moo decoding task

Additional Reading Slides

Interpretive vs compiled languages

- Python is an interpretive language.
- This means that your code is not directly run by the hardware. It is instead passed to a *virtual machine*, which is just another programme that reads and interprets your code. If your code used the '+' operation, this would be recognised by the interpreter at run time, which would then call its own internal function 'add(a,b)', which would then execute the machine code 'ADD'.
- This is in contrast to compiled languages, where your code is translated into native machine instructions, which are then directly executed by the hardware. Here, the '+' in your code would be translated directly in the 'ADD' machine code.

- Create an empty list.

```
new_list = []
```

- Use the range() and append() functions to add the integers 1-20 to the empty list.

```
for i in range(1, 21):
    new_list.append(i)
```

- Print the list to the screen, what do you have?

```
print(new_list)
```

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

Reading and writing to files in Python: The file object

- File handling in Python can easily be done with the built-in object file.
- The file object provides all of the basic functions necessary in order to manipulate files.

- Open up notepad or notepad++. Write some text and save the file to a location and with a name you'll remember.

Reading and writing to files in Python: The file object

- File handling in Python can easily be done with the built-in object file.
- The file object provides all of the basic functions necessary in order to manipulate files.

- Open up notepad or notepad++. Write some text and save the file to a location and with a name you'll remember.

The open() function

- Before you can work with a file, you first have to open it using Python's in-built open() function.
- The open() function takes two arguments; the name of the file that you wish to use and the mode for which we would like to open the file

```
practiceFile = open('Practice_file_for_IOC.txt', 'r')
```

- By default, the open() function opens a file in 'read mode'; this is what the 'r' above signifies.
- There are a number of different file opening modes. The most common are: 'r'= read, 'w'=write, 'r+'=both reading and writing, 'a'=appending.

- Use the open() function to read the file in.

Basic of Git & GitHub

RECAP: Importance

- **Python**: It's a user-friendly language for data analysis.
- **Git**: Helps manage code and data changes when working with a team.
- **Pandas**: Simplifies data cleaning and manipulation.
- **Databases**: Needed for storing and retrieving data efficiently.

These skills are essential for effective data analysis, collaboration, and handling data in real-world projects.

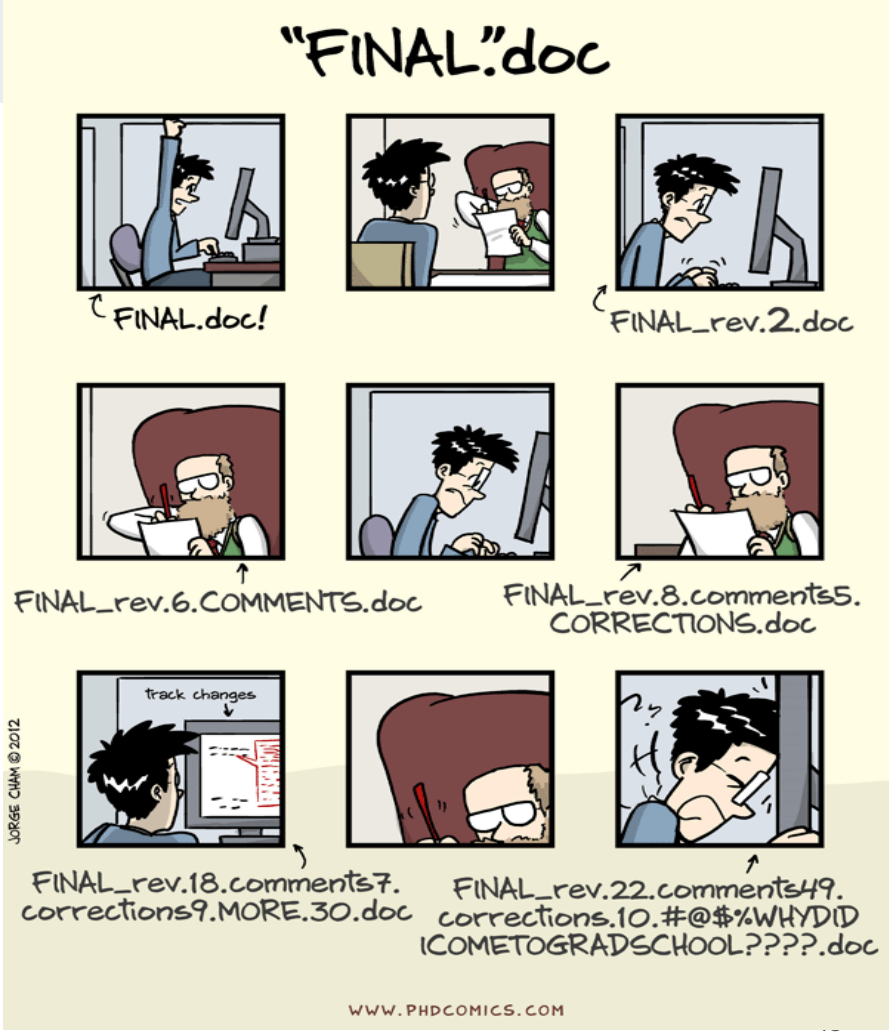
Git

Version Control System (VCS)

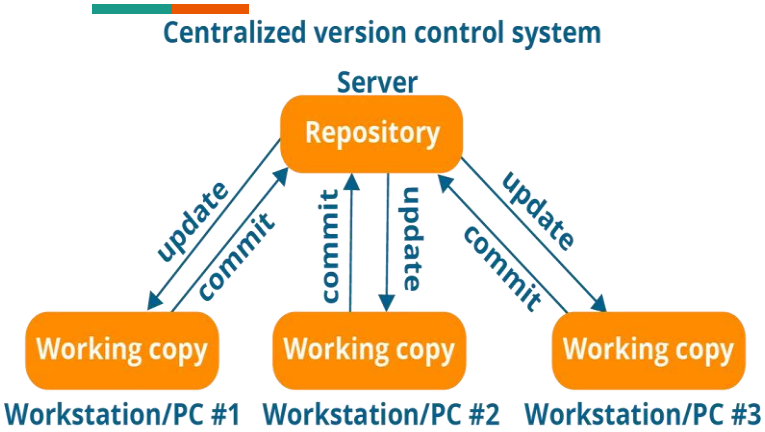
A software tool used by developers to manage changes to source code over time.

- **Tracks** modifications to files, **allowing multiple** developers to collaborate on a project simultaneously **while maintaining a complete history** of all changes made to the codebase.
- If there are issues, a version control system **allows you to revert** to an earlier code version.

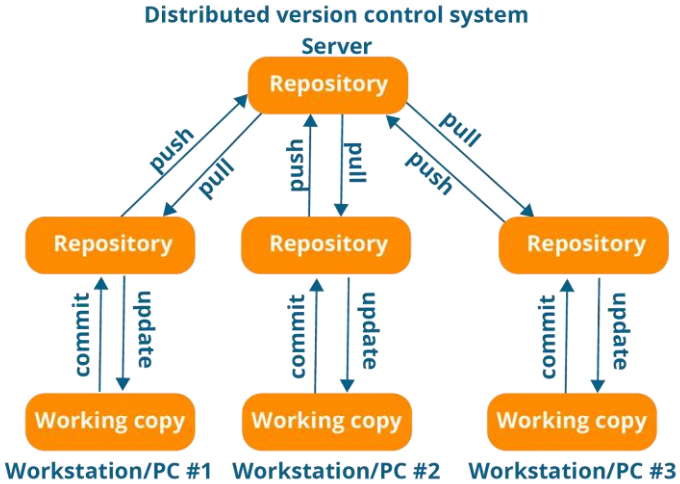
Git is the most popular version control system in the world.



VCS Types



There is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy directly.



Each programmer has a full project copy on their local machine. They commit changes locally and use push and pull operations to sync with a shared server for collaboration.

Git

Git is a **version control system** that helps developers manage and track changes in their code. It's like a time machine for your code, allowing you to go back to previous versions if something goes wrong or if you need to see how your code looked at a certain point in time.

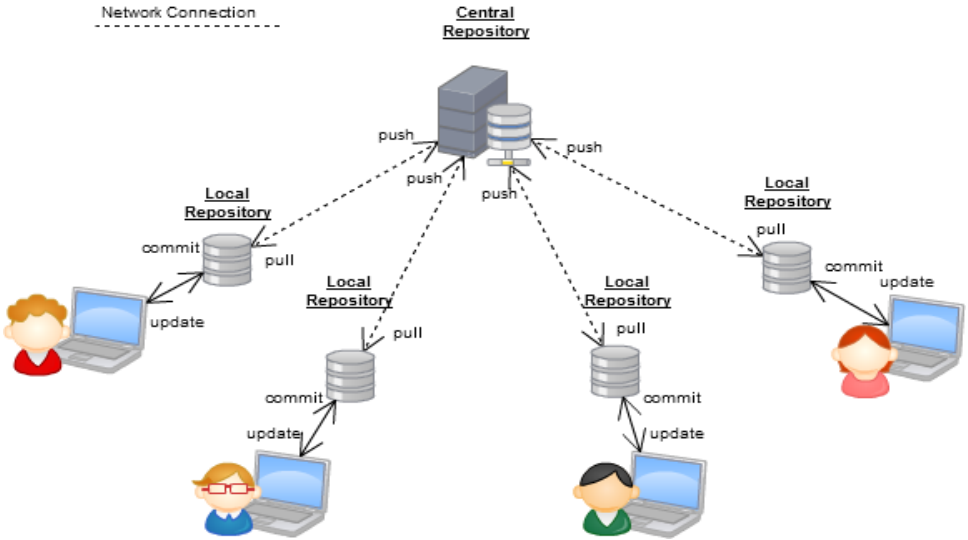
Git

A distributed version control system , used by most major corporations and labs

- **Keep a history** of previous versions
- Everyone **mirrors the entire repository** instead of simply checking out the latest version of the code
  - Unlike a centralized version control system, where the project is stored on a central server, each team member using Git has a copy of the project with its history stored on their local machine.
- **Develop simultaneously** on different branches
  - Easily try out new features, integrate them into production or throw them out
- **Collaborate** with other developers
  - “Push” and “pull” code from hosted repositories such as Github

Why Git?

- Most commonly used software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.



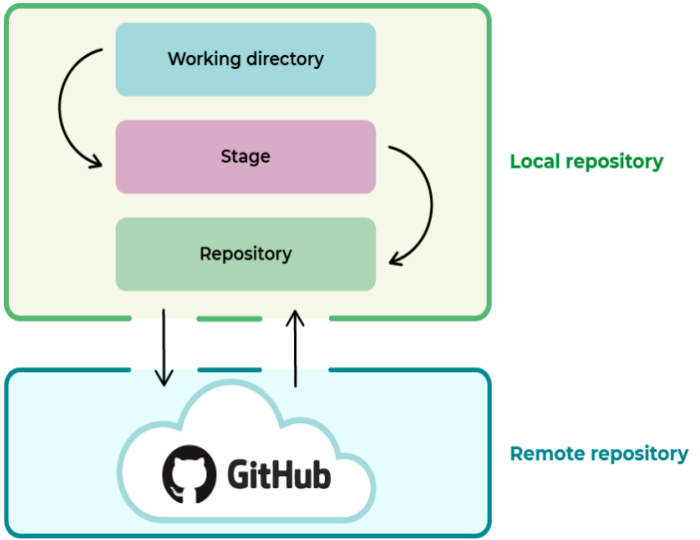
Installing Git

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Git Repository

After installing Git, you can initialize it for a project to create a new Git repository.

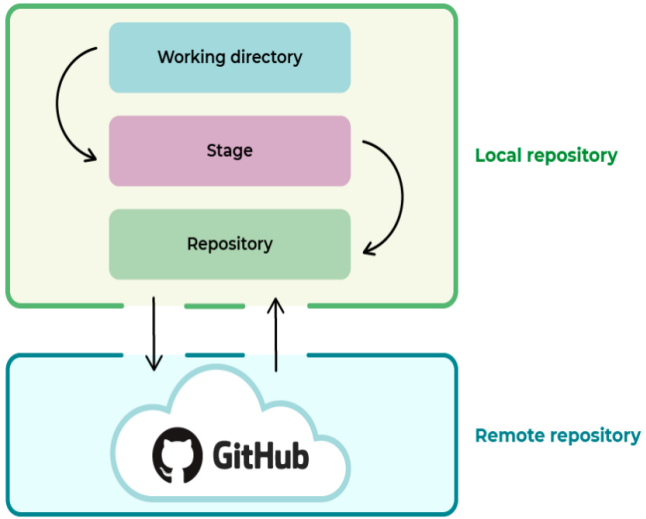
The Git Repository, represented by the '.git' directory, is a database that contains all the project source code and the history of all code updates. The '.git' directory is typically located at the root of the project directory.



Git Repository

While a Git repository can be stored on your local machine, especially for solo projects, it's common practice to store project code in a repository on a remote server.

- Git repositories may include **configuration files** such as **.gitignore** for specifying ignored files and directories, and **.gitattributes** for defining attributes like line endings or merge strategies (e.g., ensuring consistent line endings across platforms).



GitHub

GitHub is a **web-based platform** that serves as a hosting service for Git repositories.

- Provides a wide range of features and tools to facilitate collaborative software development and version control.
- Serves as a central hub for software development, enabling collaboration, code sharing, and project management for individuals, teams, and communities worldwide.

Git Commands

- ✔ git clone <project URL>
- ✔ git add my_file.c
- ✔ git commit
- ✔ git status
- ✔ git branch
- ✔ git checkout
- ✔ git push

Git Command

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

To start a new project with Git

We can either

- initialize a new repository in an existing project directory or
- clone an existing repository from a remote location (like GitHub) onto your local machine.

Git init

Initialize a new repository:
Git init creates an empty git repository for project tracking. Once the repository is created, any files added to the repository are automatically tracked.

command:

```
git init
```

Git Clone

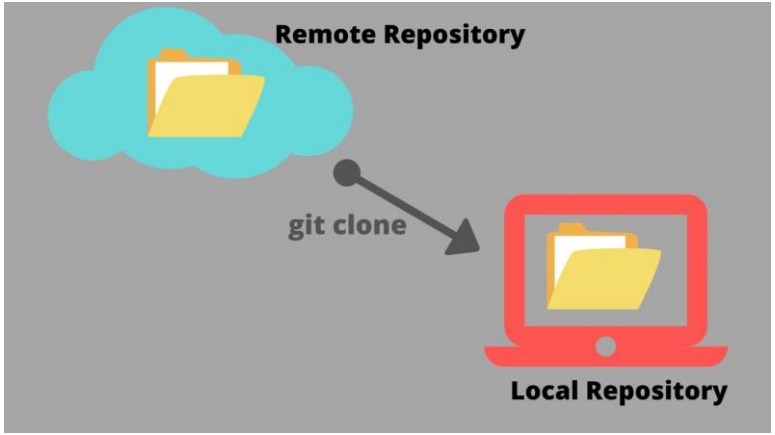
- Creates a clone/copy of an existing repository into a new directory.
  - Eg.Clones a repository from a remote location to your local machine.

```
git clone ssh://x_person@example.com/path/to/team-project.git
cd team-project
# You must work on this project
```

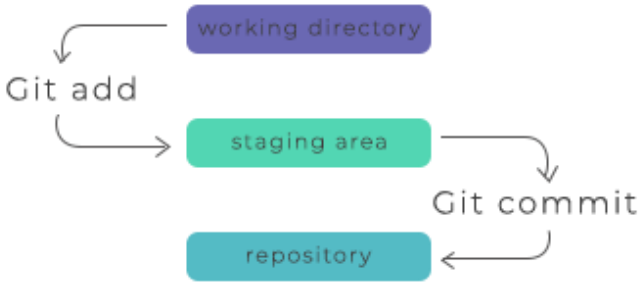
Git Clone

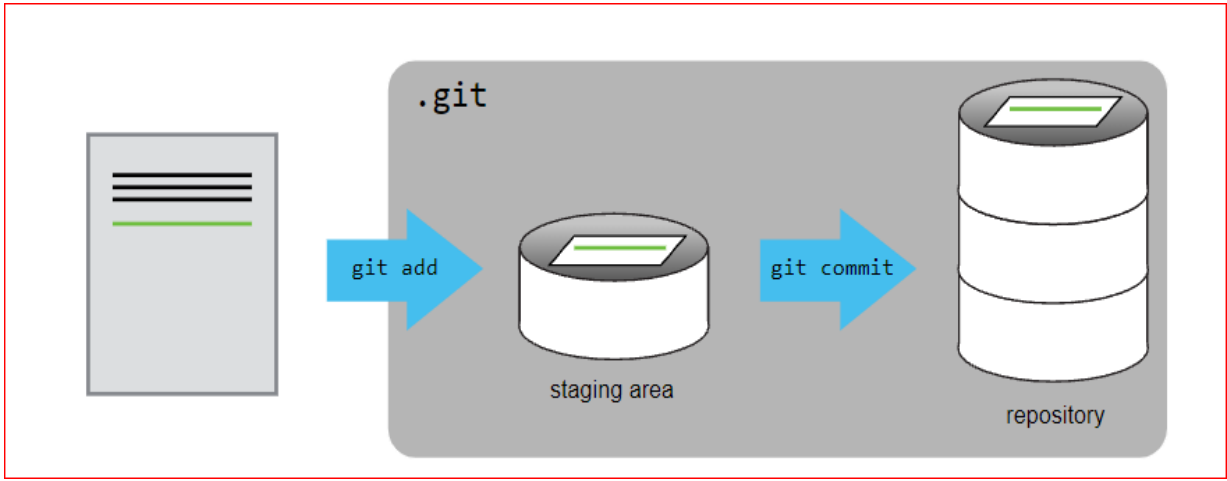
```
git clone ssh://x_person@example.com/path/to/team-project.git
cd team-project
# You must work on this project
```

- **Downloads an exact copy** of the files from the remote/central repository (hosted on a platform like GitHub, GitLab, etc.) into a newly initialized local repository on your computer.
- From there, we can check out files and make changes.
- This example establishes a new Git repository locally named 'team-project', allowing us to work on the project locally with version control.



Adding a file to git repository: git add, commit



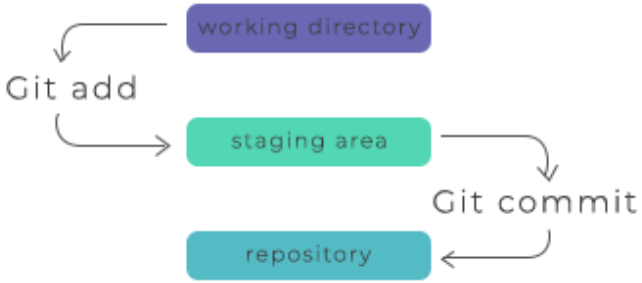


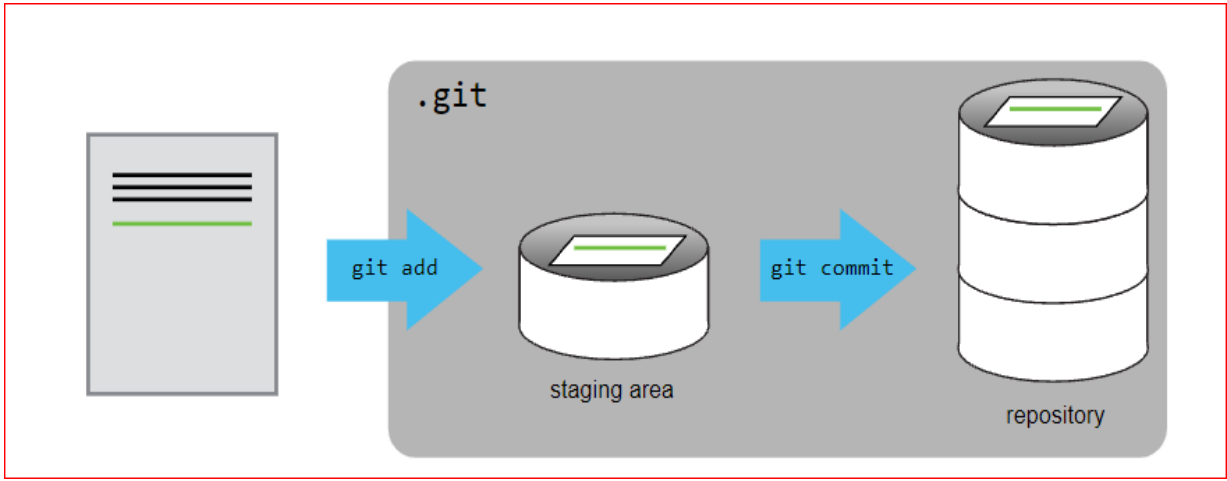
- $ cd desktop/test-GitDemo
- $ git init
- $ echo “line1” >> firstfile.txt
- $ echo “line1” >> secondfile.txt
- $ git status

git init command is used to initialize a new Git repository in a directory.

→ shows which files have been changed and need to be committed to the repository.

Adding a file to git repository: git add, commit





- $ git add >> firstfile.txt : add files to the staging staging areaOr
  $ git add >> firstfile.txt secondfile.txt
- $ git commit –m “first two files commit” : permanently storing changes in the repository.
- $ git status
- $ git log : displays a history of commits in your Git repository

Step 1: Staging telling Git to include this file in the next commit.

Step 2: Commit committed those changes with a message

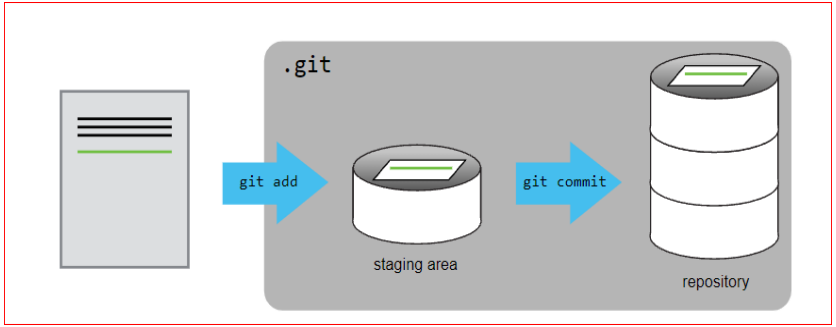
Staging is the process of organizing and preparing our project files for a commit. It is the intermediate step between modifying our files and storing them permanently in the repository.

Adding Files to Git

Git Staging Area

The staging area in Git is an intermediate area where changes are prepared before they are committed to the repository.

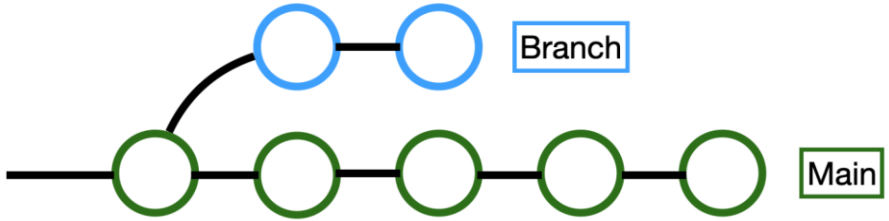
- Staging is the process of organizing and preparing our project files for a commit.
- It is the intermediate step between modifying our files and storing them permanently in the repository.



Git Branch

We can use the git branch command to

- show/list all project branches.
- create a new branch
- delete an existing branch or



Show all the branches that exist in the rep

```
git branch
```

* □ Current Branch

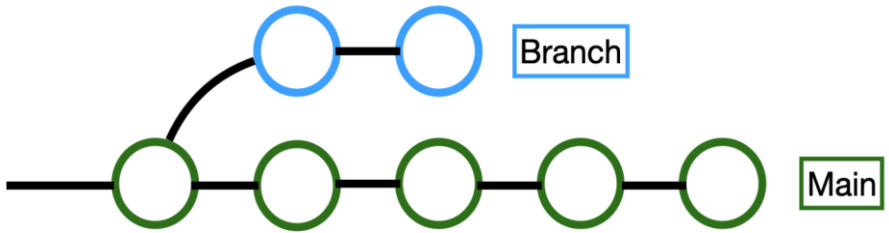
Output:

```
*master
 feature
 develop
```

Git Branch

Delete an existing branch:

git branch -d <branch name>



Create a new branch

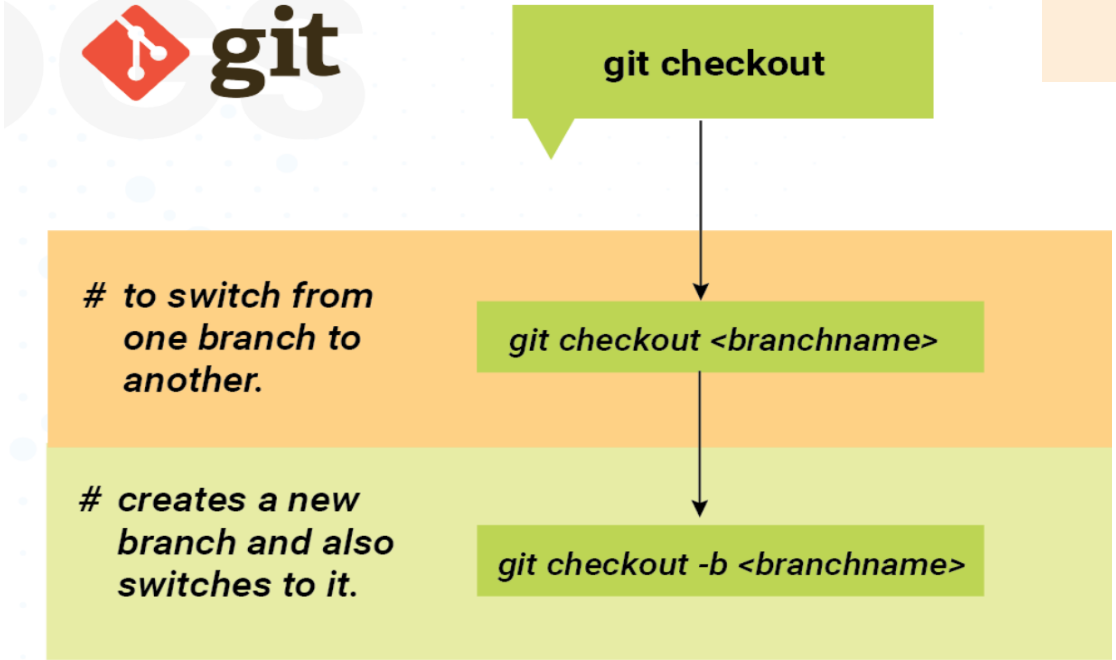
```
git branch new-branch
```

* □ Current Branch

We start with a repository, create our branch, make changes on our branch, and when we're finished, we merge those changes into the "main" branch. We ensure everything works together and then upload it back to GitHub.

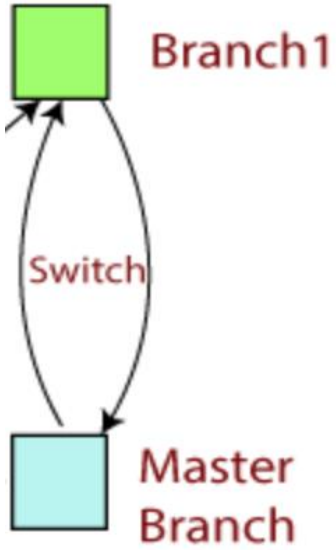
Git Checkout

Switch between branches in a repository.



REMEMBER:

- Before switching to a new branch: Changes in the existing branch must be committed.
- The new branch must exist on your local machine.

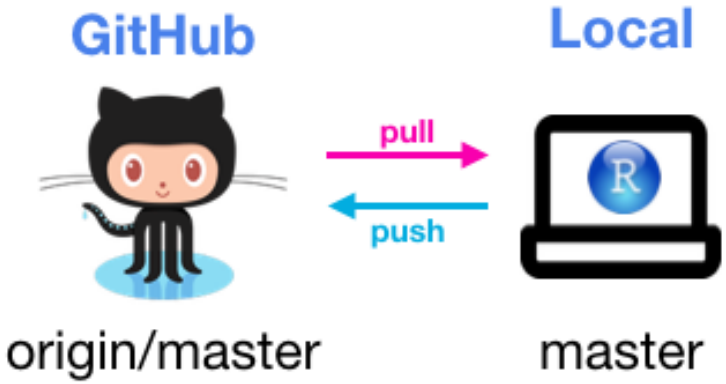


Git Push

Upload / push your local Git commits to a remote repository, such as one hosted on a platform like GitHub.

- Crucial for collaborating with others and keeping your remote repository up to date.

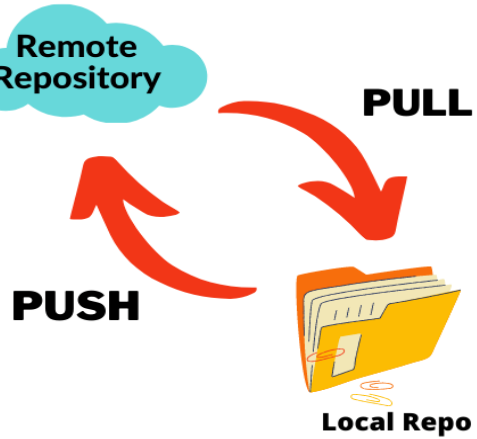
$ git push origin-main local-master



Git Pull

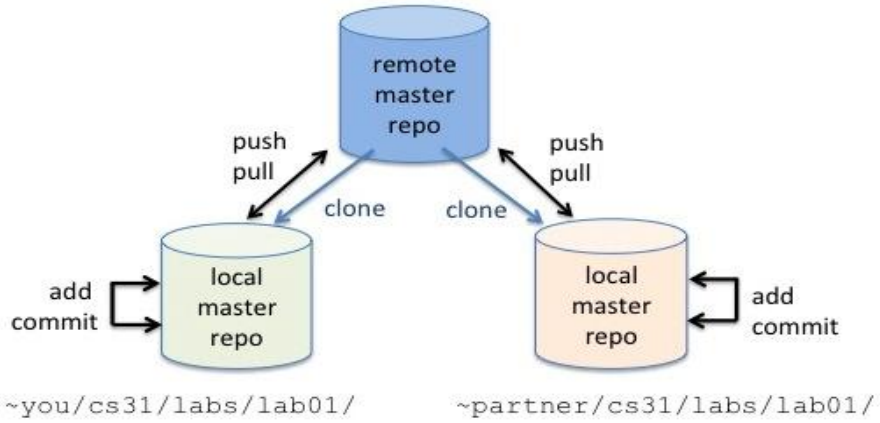
Refers to refers to fetching changes from a remote repository. Download and update our local repository with the changes made to the remote repository. First fetch the changes and then merge them into the working branch.

$ git pull <remote-repo-url>

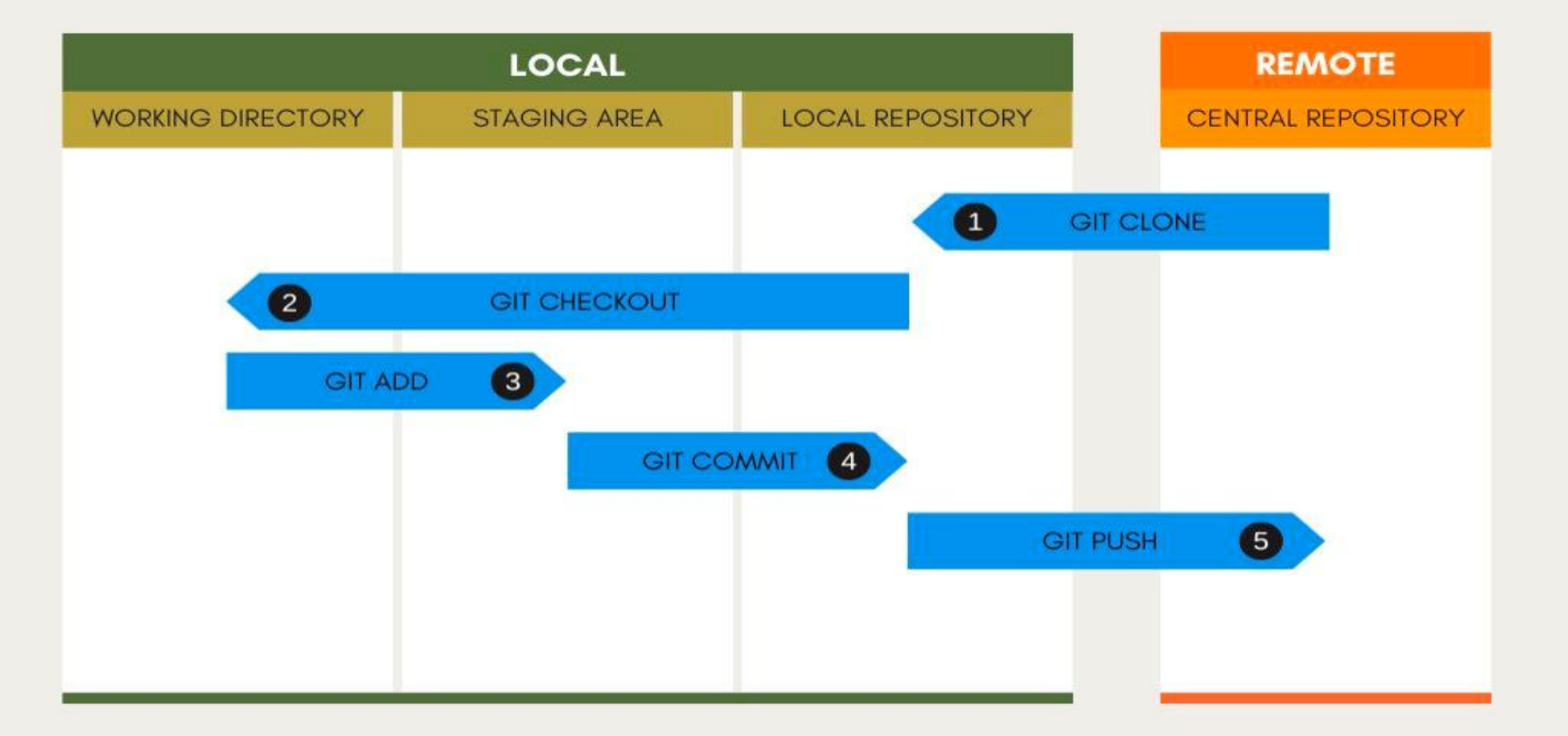


Git Overall

1. Use an existing remote master git repository
2. **Clone** a private local copy of the remote master repo
3. A**dd** and **commit** changes to your local private cloned copies.
4. Local changes can be shared others by **push**ing them to the master repo.
5. Your partners will then **pull** your pushed changes from the remote master into their local repo.



A Basic Git Workflow



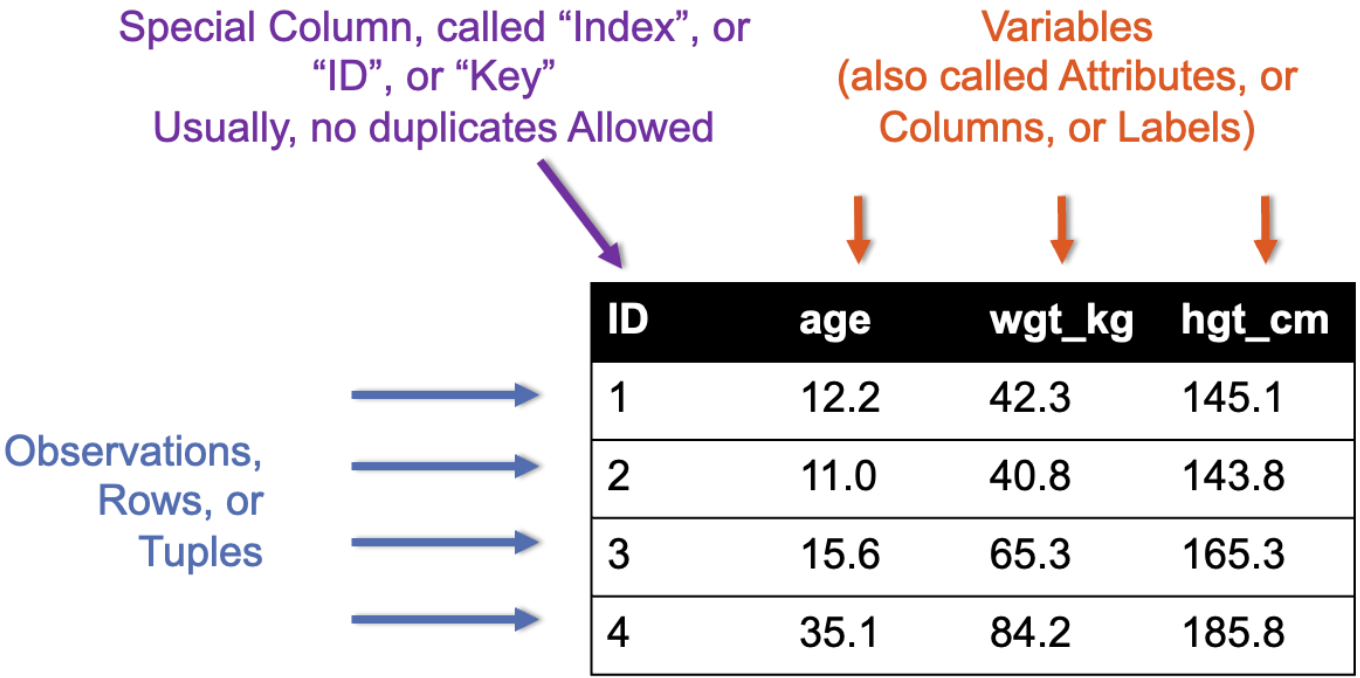
Importance

- **Python**: It's a user-friendly language for data analysis.
- **Git**: Helps manage code and data changes when working with a team.
- **Pandas**: Simplifies data cleaning and manipulation.
- **Databases**: Needed for storing and retrieving data efficiently.

These skills are essential for effective data analysis, collaboration, and handling data in real-world projects.

Basic Concept Review: Table / Tabular Data

In tabular data, information is organized into rows and columns.



ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Tabular data is crucial for both Pandas and SQL

Provides a structured and organized way to represent, manipulate and visualize data.

- Both pandas and SQL are commonly used for working with tabular data, making it essential for tasks such as data analysis, manipulation, and visualization.

Next

Pandas

Class Colab: https://colab.research.google.com/drive/1DIJXtoeaHndPiykVUJg0CUqj-RZxtj2T?usp=sharing

Dataset: https://drive.google.com/file/d/1au5qJnRR57pOeAt4aefzxwsmG02JyaR_/view?usp=sharing

Pandas: Python Library for Manipulating Tabular Data

"Pandas": Short for "Python Data Analysis Library," used for data analysis.

- Written by Wes McKinney: Started in 2008 to get a high-performance, flexible tool to perform quantitative analysis on financial data

Pandas: Python Library for Manipulating Tabular Data

"Pandas": Short for "Python Data Analysis Library," used for data analysis.

- Open source, free to use
- Analyze data locally that can fit in memory
- Almost entirely functional
- Integrates with most major frameworks in data pipelines (e.g scikit-learn, numpy etc.).
- Compatible with various data formats: CSV, Excel, XML, JSON, and relational databases.

Pandas First Steps: install and import

- Pandas is an easy package to install. Open up your terminal program (shell or cmd) and install it using either of the following commands:

```
$ conda install pandas
        OR
$ pip  install  pandas
```

- For **`jupyter notebook`** users, you can run this cell:
  - The `!` at the beginning runs cells as if they were in a terminal.

```
!pip install pandas
```

- For Google Colab: already pre-installed; just import it
- To import pandas we usually import it with a shorter name since it's used so much:

```
import pandas as pd
```


Installation: https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html

Core components of Pandas:
Series & DataFrames

Pandas: Data Table Representation

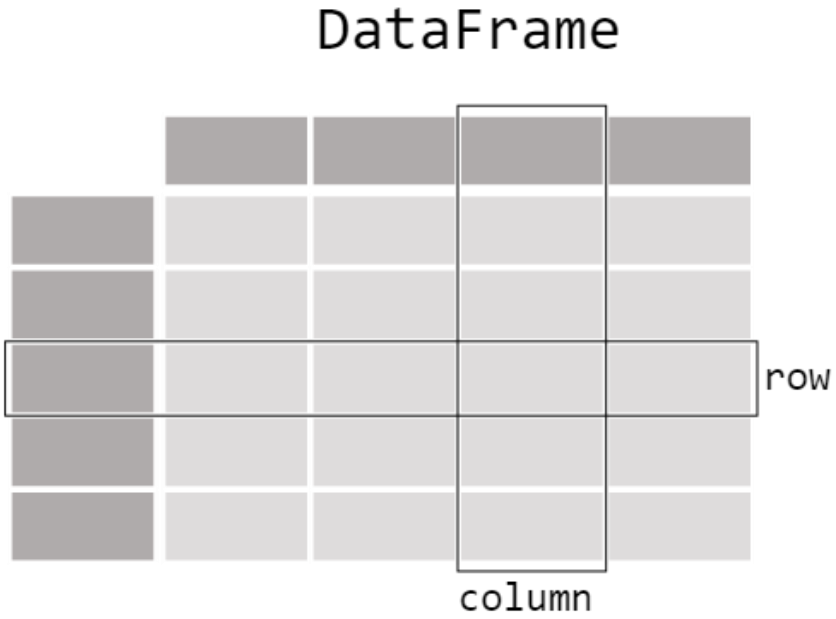
Data in Pandas is stored in two fundamental data types:

- Series is the data type that stores a single column of data.
- DataFrame is the data type that stores an entire dataset.

Notes: They both behave very similar to numpy arrays, so you can use them as input to anything in numpy, scipy, or scikit-learn. The main difference from numpy arrays is indexing.

Pandas: Data Table Representation cont.

In pandas, the most basic object is the DataFrame, that is a collection of Series objects. Each column in a DataFrame is essentially a Series.



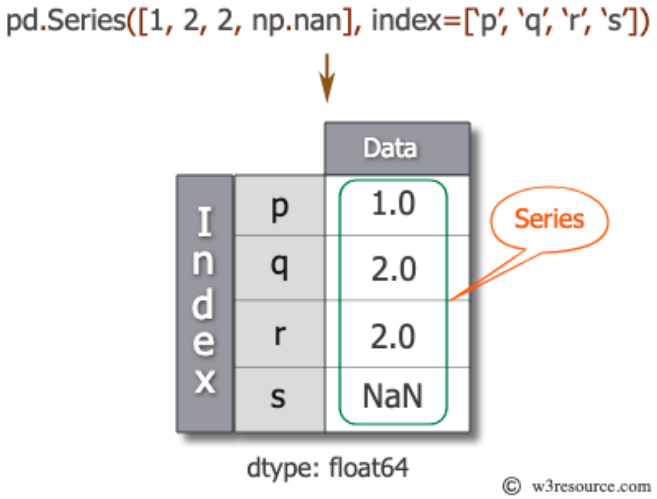
(Series)

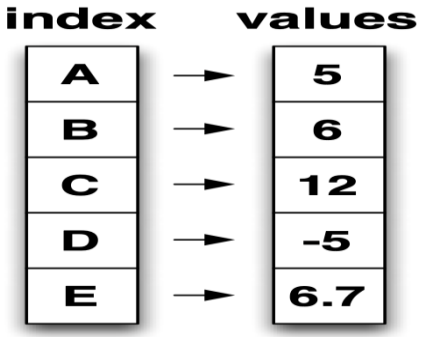
The primary two components of pandas are the Series and DataFrame.

- Series is essentially a column, and
- DataFrame is a multi-dimensional table made up of a collection of Series.

Series

- An one dimensional array of data with an index
- Subclass of numpy.ndarray
- **Data:** any type
- **Index:** most commonly integers, start with 0 (default).
    - Index labels need not be ordered
    - We almost NEVER access a series by an index
- Duplicates possible but result in reduced functionality, may produce inaccurate result.





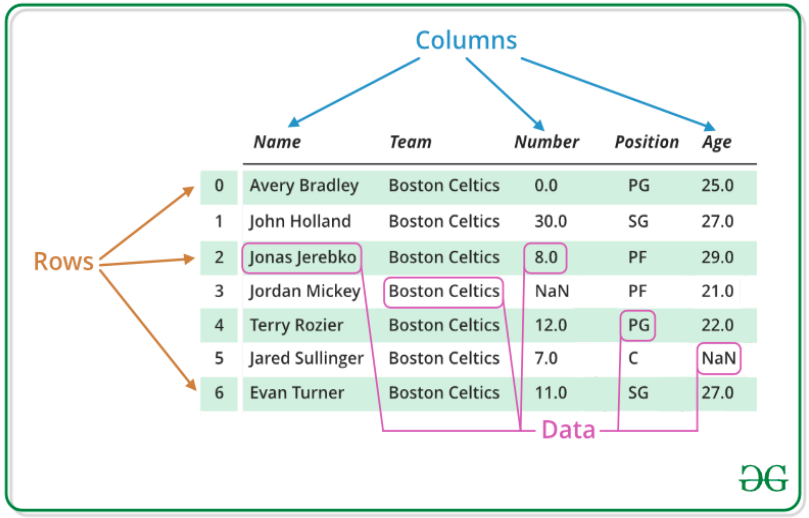
Series Functions

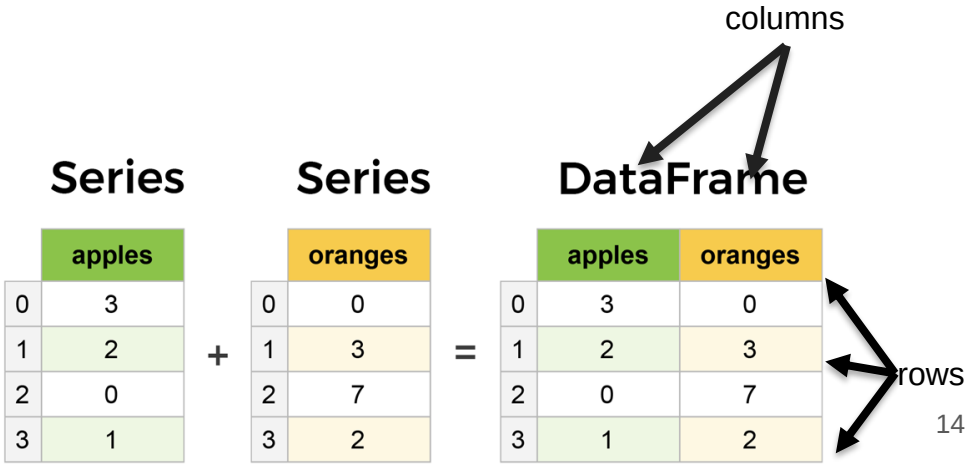
- Pandas can do anything with a series you can do with an array
  - Sum, count, show unique, etc
  - If you have a basic thing you want to do with a series, look it up!
- apply()
  - Apply takes a function as an argument and returns a series with that function applied
- You can also convert a series into a series of booleans by applying a logical condition (which will be useful later)

Dataframes

A collection of series organized by columns

- Each column can have a different type (integers, floats, strings, etc.)
- Mutable size: insert and delete columns
- Can Perform Arithmetic operations on rows and columns
- We do not access rows via index (less common).
- We DO access columns via their names



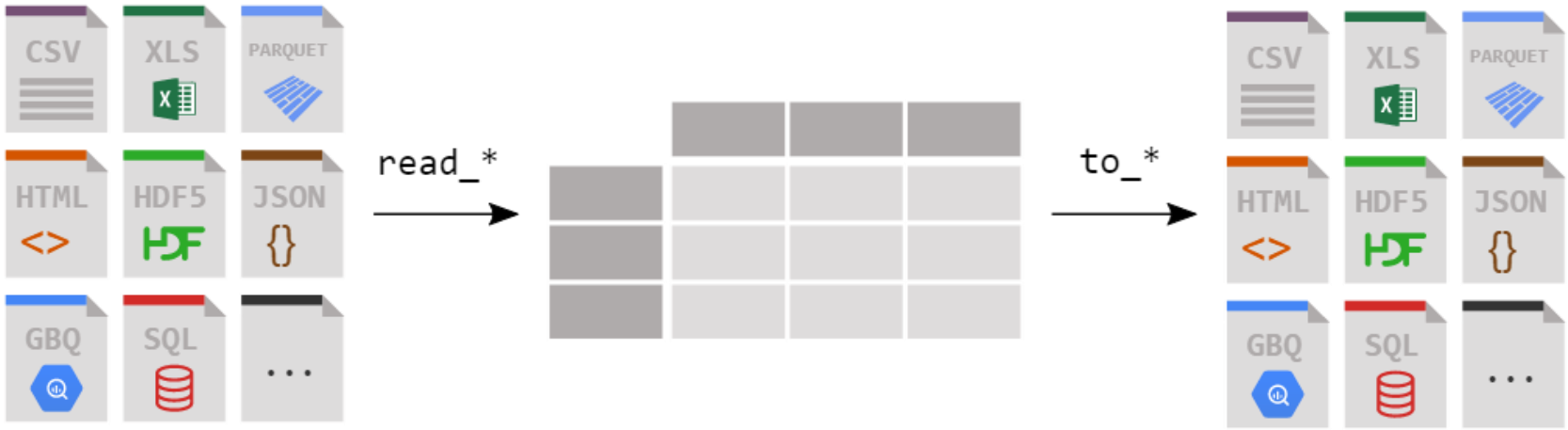


Dataframe Functions

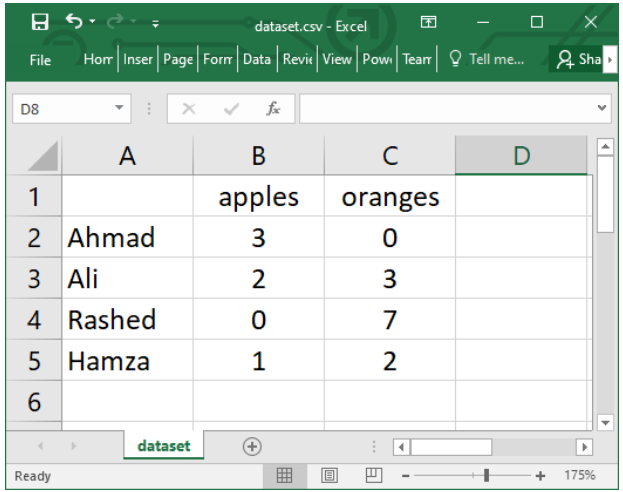
Important functions:

- How to access a column
- How to filter a column
- Apply (LATER TOPIC)
- GroupBy

Loading a DataFrame from files



Load a CSV file into a Pandas DataFrame:



	A	B	C	D
1		apples	oranges	
2	Ahmad	3	0	
3	Ali	2	3	
4	Rashed	0	7	
5	Hamza	1	2	
6				

Reading data from a CSV file

File Edit Format Run Options Window Help

```
1 import pandas as pd
2
3 df = pd.read_csv('dataset.csv')
4 print(df)
5
```

- With CSV files, all you need is a single line to load in the data:

```
df = pd.read_csv('dataset.csv')
```

	Unnamed: 0	apples	oranges
0	Ahmad	3	0
1	Ali	2	3
2	Rashed	0	7
3	Hamza	1	2

Viewing data

Use DataFrame.head() and DataFrame.tail() to view the top (first) and bottom (last) few rows of the frame respectively.

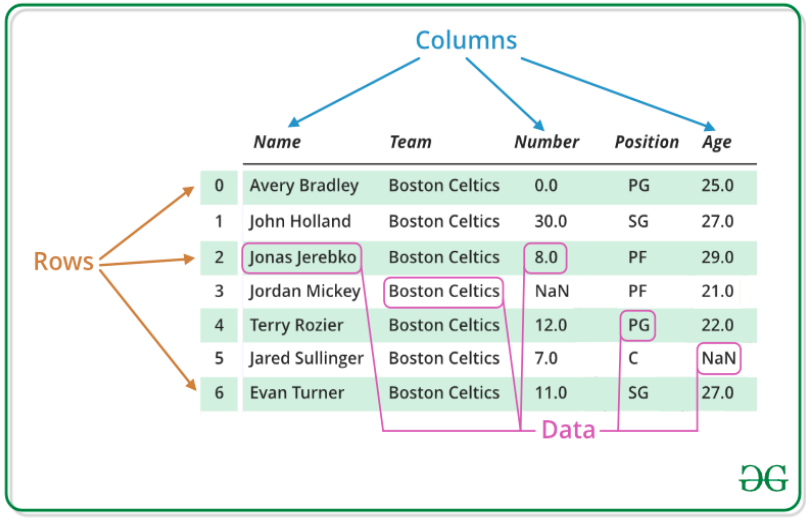
- Default value is 5, unless specified otherwise.

How to access a column

dataframe['column_name']: Access a column by its name.

E.g. : df['age']

Where df=your dataframe



Arithmetic Operations

Can perform arithmetic operations (e.g., addition, subtraction, multiplication, division) on entire columns or between columns.

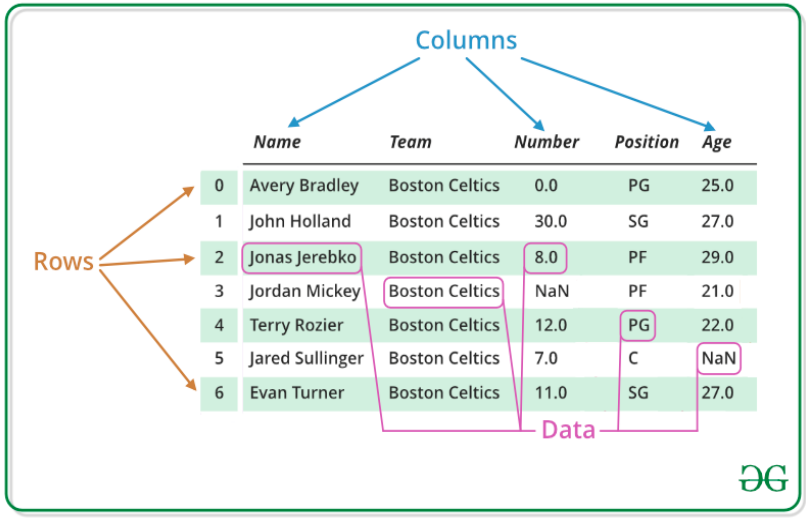
E.g.:

```
df['new_column'] = df['column1'] + df['column2']
df['new_column'] = df['column1'] - df['column2']
```

Applying Functions Directly to the dataframe

You can apply different functions directly to a DataFrame column:

Syntax: **dataframe[column_name].function()**

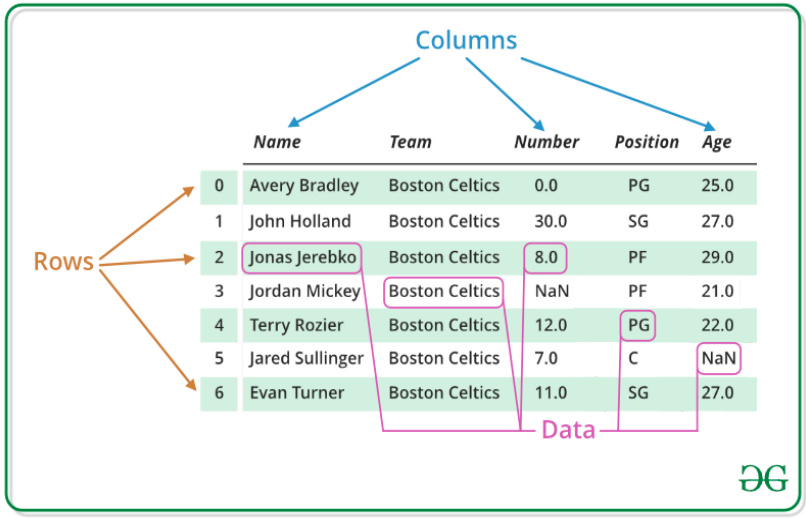


Applying Functions Directly to the dataframe

Aggregation operations are used to compute summary statistics or single values from multiple values in a dataset. E.g:

E.g.: df[“age”].sum()

Try some other aggregation operation!



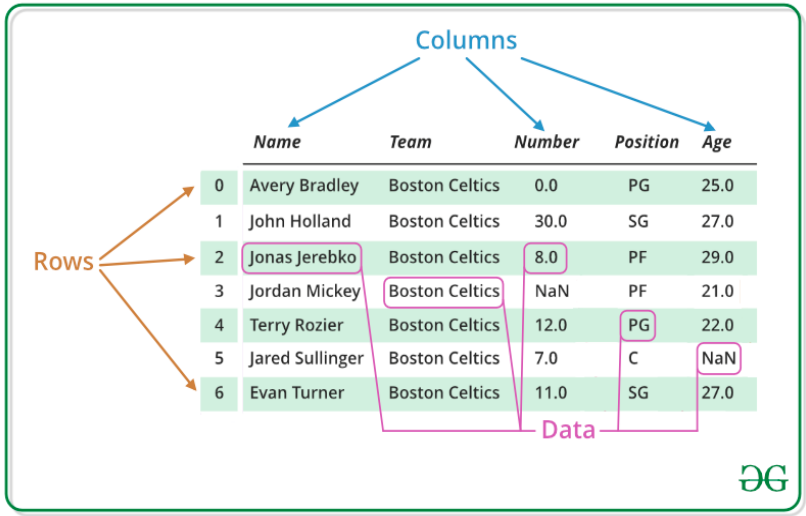
How to Filter a Column based on Condition (s)

To filter a column in a pandas DataFrame, we can use boolean indexing.

E.g.: Filter only the 'Name' column for values equal to 'Jonas Jerebko'

dataframe[‘column name’][boolean condition]

df[‘Name’][df[‘Name’]=='Jonas Jerebko']



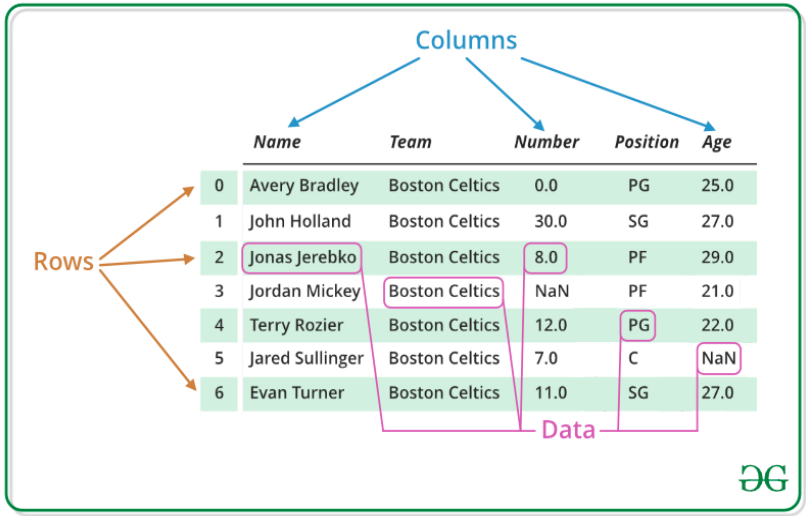
How to Filter Rows based on Condition (s)

E.g.: Filter rows where age is greater than 25.

dataframe [boolean condition]

df [df['Age']>29]

Q: What about multiple condition?



More Advanced: Filtering Data & Apply Statistical Functions

We can filter rows in a DataFrame based on a condition and then apply statistical functions to a specific column:

```
df[df['condition']][column_name].statistics_function()
```

- **Filters the DataFrame** based on the condition specified within the brackets.
- It **selects only the rows** that satisfy the condition.

- Selects the **specific column** from the filtered DataFrame obtained in left, resulting in a Series containing only the values from that column.

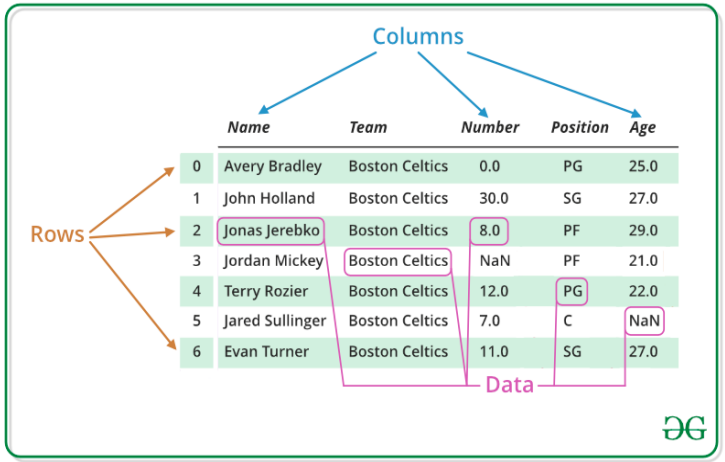
- **Applies** a statistical function to the Series obtained in the left to compute a summary statistic.

More Advanced: Filtering Data & Apply Statistical Functions

We can filter rows in a DataFrame based on a condition and then apply statistical functions to a specific column:

df[df['condition']][column_name].statistics_function()

Try: Calculate the mean of the 'Number' column where the 'Age' is greater than 25



Grouping Data (Groupby)

You can group data by a condition and calculate statistics within each group.

- This process is commonly referred to as "split-apply-combine."

```
df.groupby('condition')[column_name].statistics_function()
```

Splitting

- Groups the DataFrame based on the values in the 'condition' column.
- **Splits the DataFrame** into groups based on unique (identical) values in the specified column.

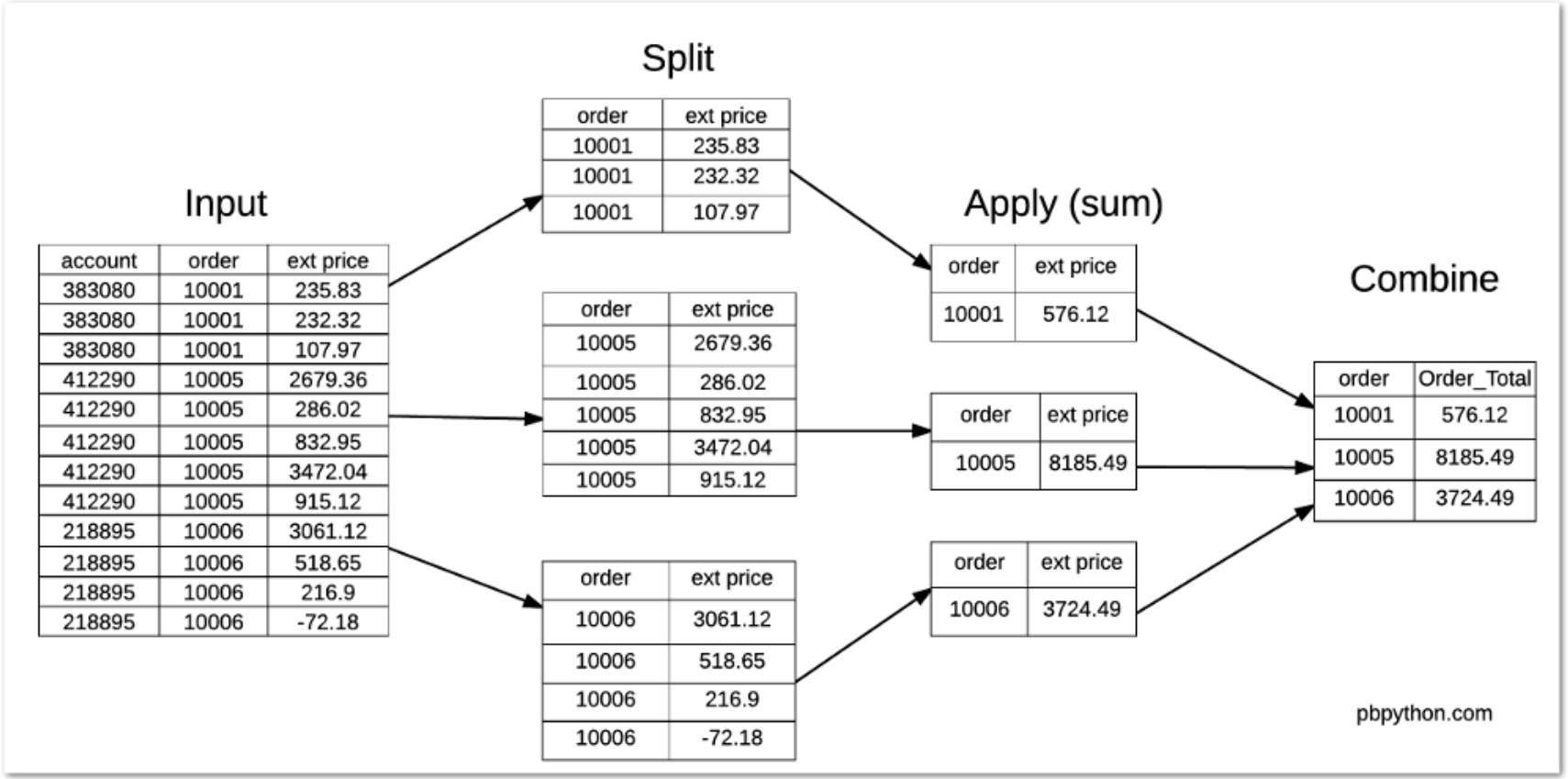
Apply

- Selects a specific column from each group, resulting in a Series containing the values from that column within each group..

Combine

- **Applies** a statistical function to each group of values in the selected column.

Grouping Data (Groupby)



Write data to a CSV (Comma Separated Values) file

For a Series s: s.to_csv("filename.csv")

For a DataFrame df: df.to_csv("filename.csv")

Pandas: Advantages

- A powerful python library for data tasks like analysis and cleaning.
- Simplifies data representation for better understanding.
- Cleans messy datasets for readability and relevance.
- Increases productivity by minimizing writing.
- Offers extensive features for easy data analysis.

References

- pandas documentation
  - https://pandas.pydata.org/pandas-docs/stable/index.html
- pandas: Input/output
  - https://pandas.pydata.org/pandas-docs/stable/reference/io.html
- pandas: DataFrame
  - https://pandas.pydata.org/pandas-docs/stable/reference/frame.html
- pandas: Series
  - https://pandas.pydata.org/pandas-docs/stable/reference/series.html
- pandas: Plotting
  - https://pandas.pydata.org/pandas-docs/stable/reference/plotting.html