bicycles = ['trek', 'cannondale', 'redline', 'specialized']

print(bicycles[0].title())

Declaring a list

Bicycles = ['trek', 'Cannondale', 'redline', 'specialized']

Print a list by passing it to print.  
Bicycles = ['trek', 'Cannondale', 'redline', 'specialized']

print(bicycles)

Printing the first item from a list.  
bicycles = ['trek', 'cannondale', 'redline', 'specialized']

print(bicycles[0])

Print the last item in the list.  
bicycles = ['trek', 'cannondale', 'redline', 'specialized']

print(bicycles[-1])

Print the second last item in the list.  
bicycles = ['trek', 'cannondale', 'redline', 'specialized']

print(bicycles[-2])

Use f-string to create a message having a value from a list  
bicycles = ['trek', 'cannondale', 'redline', 'specialized']

message = f"My first bicycle was a {bicycles[0].title()}."  
print(message)

Modifying an element in a list  
motorcycles = ['honda', 'yamaha', 'suzuki']

motorcycles[0] = 'ducati'

Appending to the end of a list with existing items  
motorcycles = ['honda', 'yamaha', 'suzuki']

motorcycles.append('ducati')

# Original list

original\_list = [1, 2, 3, 4, 5]

new\_list = original\_list + [6, 7]

[1, 2, 10, 3, 4, 5]

Declaring an empty list

motorcycles = []

Add an element at position 0  
motofrcycles = ['honda', 'yamaha', 'suzuki']

motorcycles.insert(0,'ducati')

Removing an item from a list using its index position.  
The index position is 1  
motorcycles = ['honda', 'yamaha', 'suzuki']

del motorcycles[1]

Remove the last item from a list.  
Use the value of the item after removing it from the list  
motorcycles = ['honda', 'yamaha', 'suzuki']

last\_owned = motorcycles.pop()  
print(f"The last motorcycle I owned was a {last\_owned.title()}")

Remove the item from the list with index position 0.  
Use the value after removing it from the list  
motorcycles = ['honda', 'yamaha', 'suzuki']

first\_owned = motorcycles.pop(0)  
print(f"The first motorcycle I owned was a {first\_owned.title()}")

Remove item 'ducati' by value  
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']

motorcycles.remove('ducati')

Sorting a list permanently in alphabetical order  
cars = ['bmw', 'audi', 'toyota', 'subaru']

cars.sort()

Sorting in reverse alphabetical order.  
Changing the order permanently  
cars = ['bmw', 'audi', 'toyota', 'subaru']

cars.sort(reverse=True)

**Sort by Length of Strings**

cars = ['bmw', 'audi', 'toyota', 'subaru']

cars.sort(key=len)

print(cars)

Sorting a list temporarily in alphabetical order.  
The list is not modified permanently.  
cars = ['bmw', 'audi', 'toyota', 'subaru']

print(sorted(cars))

Sorting a list temporarily in reverse alphabetical order.  
The list is not modified permanently.  
cars = ['bmw', 'audi', 'toyota', 'subaru']

print(sorted(cars, reverse=True))

Note to sort upper case and lower case string

# List with mixed case letters

words = ["apple", "Banana", "cherry", "Date", "elderberry", "Fig", "grape"]

# Default sorting

sorted\_default = sorted(words)

# Case-insensitive sorting

sorted\_case\_insensitive = sorted(words, key=lambda s: s.lower())

print("Default sorting:", sorted\_default)

print("Case-insensitive sorting:", sorted\_case\_insensitive)

Output:

python

Copy code

Default sorting: ['Banana', 'Date', 'Fig', 'apple', 'cherry', 'elderberry', 'grape']

Case-insensitive sorting: ['apple', 'Banana', 'cherry', 'Date', 'elderberry', 'Fig', 'grape']

Printing a list in reverse order.  
The order of the list is changed permanently.  
The elements are not sorted backward alphabetically.  
The order is just reversed.  
cars = ['bmw', 'audi', 'toyota', 'subaru']

cars.reverse()  
print(cars)

Length of a list  
cars = ['bmw', 'audi', 'toyota', 'subaru']

print(len(cars))

We get IndexError.  
motorcycles = ['honda', 'yamaha', 'suzuki']

print(motorcycles[3])

Looping through an entire list printing each item individually  
magicians = ['alice', 'david', 'carolina']

for magician in magicians:  
print(magician)

We get a SyntaxError: invalid syntax as we forgot a colon.  
magicians = ['alice', 'david', 'carolina']

for magician in magicians  
print(magician)

Printing a list with items as numbers from 1 to 5.

print(list(range(1, 6))

Print even numbers

print(list(range(2, 11, 2)))

Add the first 10 square numbers to a list  
squares = []

for value in range(1,11):  
squares.append(value\*\*2)

Find minimum, maximum and sum of a list of numbers  
digits = [1,2,3,4,5,6,7,8,9,0]

print(min(digits))  
print(max(digits))  
print(sum(digits))

List comprehensions, generating a list with one line of code

squares = [value\*\*2 for value in range(1, 11)]

Print the first three items in a list  
players = ['charles', 'martina', 'michael', 'florence', 'eli']

for player in players[:3]:  
print(player.title())

Print the second, third and fourth items in a list  
players = ['charles', 'martina', 'michael', 'florence', 'eli']

print(players[1:4])

Print the last three players from a list of players  
players = ['charles', 'martina', 'michael', 'florence', 'eli']

print(players[-3:])

Print a list starting from the third item through the last item  
players = ['charles', 'martina', 'michael', 'florence', 'eli']

print(players[2:])

Copy a list to create a separate list  
my\_foods = ['pizza', 'falafel', 'carrot cake']

friend\_foods = my\_foods[:]

This code does not make a copy of a list.  
The variables point to the same list.  
my\_foods = ['pizza', 'falafel', 'carrot cake']

friend\_foods = my\_foods

Print the items of a tuple  
dimensions = (200, 50)

for dimension in dimensions:  
print(dimension)

We get a TypeError: 'tuple' object does not support item assignment  
dimensions = (200, 50)

dimensions[0]=250

A tuple with a single element

my\_t=(3,)

-------------------------------------------------------------------------------------

Note this brother

names = ['oly','eyob','zeki']

for name in names:

    print("Hello dear " + name)

for i in range(3):

    print("Hello dear " + name[i])

OUT PUT

Hello dear oly

Hello dear eyob

Hello dear zeki

Hello dear z

Hello dear e

Hello dear k

# knowing pop that it takes out but it’s not delete at first

my\_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Remove and return the value for the key 'age'

age = my\_dict.pop('age')

print(age) # Output: 30

print(my\_dict) # Output: {'name': 'Alice', 'city': 'New York'}

# Remove and return the value for a key with a default value

my\_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}

city = my\_dict.pop('city', 'Unknown')

print(city) # Output: 'New York'

print(my\_dict) # Output: {'name': 'Alice'}

-------------------------------------------------------------------------------------

**Different ways of modifying list in python**

**1. Direct Assignment**

my\_list = [1, 2, 3, 4, 5]

my\_list[2] = 10

print(my\_list)

Output: [1, 2, 10, 4, 5]

**2. Using Slice Assignment**

my\_list = [1, 2, 3, 4, 5]

my\_list[1:4] = [6, 7, 8]

print(my\_list)

Output: [1, 6, 7, 8, 5]

**3. Append and Extend Methods**

* **append()**: Adds a single element to the end of the list.

my\_list = [1, 2, 3]

my\_list.append(4)

print(my\_list)

Output: [1, 2, 3, 4]

* **extend()**: Extends the list by appending elements from an iterable (e.g., another list).

my\_list = [1, 2, 3]

my\_list.extend([5, 6])

print(my\_list)

Output: [1, 2, 3, 4, 5, 6]

**4. Insert Method**

my\_list = [1, 2, 3, 4, 5]

# Insert element 10 at index 2

my\_list.insert(2, 10)

print(my\_list)

Output: [1, 2, 10, 3, 4, 5]

**5. List Comprehension**

# Example: Multiply each element in the list by 2

my\_list = [1, 2, 3, 4, 5]

my\_list = [x \* 2 for x in my\_list]

print(my\_list)

Output: [2, 4, 6, 8, 10]

**6. Pop and Remove Methods**

* **pop(): Removes and returns the element at a specific index (or the last element if no index is specified).**

my\_list = [1, 2, 3, 4, 5]

# Remove element at index 2

my\_list.pop(2)

print(my\_list)

Output: [1, 2, 4, 5]

* **remove(): Removes the first occurrence of a specified value.**

my\_list = [1, 2, 3, 4, 5]

# Remove element with value 4

my\_list.remove(4)

print(my\_list)

Output: [1, 2, 5]

**Different ways of Adding Elements to a List**

**1. Using append() Method**

Adds a single element to the end of the list.

my\_list = [1, 2, 3]

my\_list.append(4)

print(my\_list)

Output: [1, 2, 3, 4]

**2. Using extend() Method**

Extends the list by appending elements from an iterable (e.g., another list).

my\_list = [1, 2, 3]

my\_list.extend([4, 5, 6])

print(my\_list)

Output: [1, 2, 3, 4, 5, 6]

**3. Using insert() Method**

Inserts an element at a specified index. All existing elements from that index onward are shifted one position to the right.

my\_list = [1, 2, 3]

my\_list.insert(1, 4)

print(my\_list)

Output: [1, 4, 2, 3]

**4. Using List Concatenation**

Combines two or more lists into one. This creates a new list.

my\_list = [1, 2, 3]

new\_list = my\_list + [4, 5, 6]

print(new\_list)

Output: [1, 2, 3, 4, 5, 6]

**5. Using List Multiplication**

Repeats the list a specified number of times. This creates a new list.

my\_list = [1, 2, 3]

repeated\_list = my\_list \* 2

print(repeated\_list)

Output: [1, 2, 3, 1, 2, 3]

**6. Using List Comprehension**

Creates a new list by applying an expression to each element in an existing iterable or applying conditional logic.

my\_list = [1, 2, 3]

new\_list = [x \* 2 for x in my\_list]

print(new\_list)

Output: [2, 4, 6]

**7. Using + Operator with List Concatenation**

Combines multiple lists or elements into a single list.

my\_list = [1, 2, 3]

my\_list = my\_list + [4]

print(my\_list)

Output: [1, 2, 3, 4]

**8. Using \* Operator for Repetition**

Creates a new list by repeating the elements of the list.

my\_list = [1, 2]

extended\_list = my\_list \* 3

print(extended\_list)

Output: [1, 2, 1, 2, 1, 2]

**9. Using append() with a List**

append() adds a list as a single element, rather than adding each element individually.

my\_list = [1, 2, 3]

my\_list.append([4, 5])

print(my\_list)

Output: [1, 2, 3, [4, 5]]

**CHAPTER 4**

**Diffrent Ways of Looping Through An Entire List**

**1. Using a for Loop**

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

**2.** Using **enumerate()**

enumerate() is useful when you need both the index and the value of each item in the list.

fruits = ["apple", "banana", "cherry"]

for index, fruit in enumerate(fruits):

print(f"Index {index}: {fruit}")

#OutPut

Index 0: apple

Index 1: banana

Index 2: cherry

**3.** Using a **while** Loop

You can use a while loop with an index to iterate through the list.

fruits = ["apple", "banana", "cherry"]

index = 0

while index < len(fruits):

print(fruits[index])

index += 1

**4.** Using List Comprehensions

List comprehensions are a concise way to loop through a list and apply an operation.

fruits = ["apple", "banana", "cherry"]

# Create a new list with the same elements

new\_fruits = [fruit for fruit in fruits]

print(new\_fruits)

**5.** Using **map()**

map() applies a function to every item in the list and returns a map object (which can be converted to a list).

fruits = ["apple", "banana", "cherry"]

def print\_fruit(fruit):

print(fruit)

# Apply the function to each item

list(map(print\_fruit, fruits))

**6.** Using **filter()**

filter() applies a function to filter elements from the list based on a condition.

fruits = ["apple", "banana", "cherry"]

def starts\_with\_b(fruit):

return fruit.startswith('b')

# Filter the list based on the condition

b\_fruits = list(filter(starts\_with\_b, fruits))

print(b\_fruits)

**7.** Using a **for** Loop with **range()**

If you need to work with indices, you can use range() in a for loop.

fruits = ["apple", "banana", "cherry"]

for i in range(len(fruits)):

print(fruits[i])

**8.** Using **itertools**

The itertools module provides functions that can be used to iterate over lists.

import itertools

fruits = ["apple", "banana", "cherry"]

for fruit in itertools.cycle(fruits):

# This will loop indefinitely; to stop, you can use a break condition

print(fruit)

break # Remove this break to loop indefinitely

**Git**

Sdf

### 1. ****Set Up Git (if you haven’t already)****

Before you start, you need to configure Git with your name and email. Open your terminal (or Git Bash on Windows) and run the following commands:

bash

Copy code

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

### 2. ****Initialize a Local Repository****

Navigate to your project directory in the terminal and initialize a Git repository if you haven’t already done so:

bash

Copy code

cd path/to/your/project

git init

### 3. ****Add Your Files to the Repository****

Add the files you want to track to the staging area:

bash

Copy code

git add .

The . adds all files in the directory. You can also add specific files by replacing . with the file names.

### 4. ****Commit Your Changes****

Create a commit with a descriptive message:

bash

Copy code

git commit -m "Initial commit message"

Replace "Initial commit message" with a meaningful description of the changes.

### 5. ****Create a Remote Repository****

If you don’t have a remote repository set up yet, you’ll need to create one on a platform like GitHub, GitLab, or Bitbucket.

#### ****On GitHub****:

1. Go to [GitHub](https://github.com/).
2. Click the + icon in the upper-right corner and select **New repository**.
3. Enter a repository name and description.
4. Click **Create repository**.

### 6. ****Add the Remote Repository****

Once your remote repository is created, link your local repository to the remote one. Replace REMOTE\_URL with the URL of your remote repository, which you can find on the repository page on GitHub.

bash

Copy code

git remote add origin REMOTE\_URL

For example:

bash

Copy code

git remote add origin https://github.com/yourusername/your-repository.git

### 7. ****Push Your Code to the Remote Repository****

Push your local commits to the remote repository:

bash

Copy code

git push -u origin main

Replace main with master if your branch is named master (older Git repositories use master as the default branch name).

### 8. ****Verify Your Push****

You can verify that your code has been pushed by visiting your repository page on GitHub or the platform you used. You should see your files and commits there.

### Additional Commands

* **Check Status**: To see the status of your files and which ones are staged or modified:

bash

Copy code

git status

* **View Commit History**: To view the commit history:

bash

Copy code

git log

* **Pull Changes**: To pull changes from the remote repository to your local repository:

bash

Copy code

git pull origin main

### Summary

1. **Initialize Git**: git init
2. **Add Files**: git add .
3. **Commit Changes**: git commit -m "message"
4. **Add Remote**: git remote add origin REMOTE\_URL
5. **Push to Remote**: git push -u origin main

My Questions

should i have to specify exact place where .py or my exact project is found or simply one folder up

ChatGPT

When working with Git, it's important to understand the concept of the working directory and how paths are managed. Here’s a breakdown of how to handle paths in relation to your .py project and the Git repository:

### ****1. Initializing a Git Repository****

* **Exact Directory**: You should run git init in the root directory of your project where your .py files are located. This sets up the Git repository in that directory.

bash

Copy code

cd path/to/your/project

git init

This will create a .git directory in the path/to/your/project, which Git uses to track changes.

* **One Folder Up**: If you initialize Git one folder up from where your .py files are located, Git will track everything in that folder and its subfolders. This might be useful if your project is in a subdirectory, and you want to track multiple projects or directories.

For example, if your project structure is:

Copy code

my\_projects/

└── my\_python\_project/

├── file1.py

├── file2.py

└── subfolder/

└── file3.py

And you want to initialize Git at my\_projects to include everything under my\_python\_project:

bash

Copy code

cd path/to/my\_projects

git init

This will initialize Git at my\_projects, and all files and folders within my\_projects will be tracked, including my\_python\_project.

### ****2. Adding and Committing Files****

When adding and committing files, you should be in the directory where you initialized Git, but you don’t need to specify exact paths to the files unless you want to add files from different directories. You can use relative paths to specify files.

* **Add All Files**: If you’re in the root directory of the repository, use:

bash

Copy code

git add .

This adds all files in the current directory and its subdirectories to the staging area.

* **Add Specific Files**: If you need to add files from specific locations, you can use:

bash

Copy code

git add path/to/your/file.py

This adds the specified file to the staging area.

### ****3. Example Workflow****

Here’s a typical workflow when working with a .py project:

1. **Navigate to Your Project Directory**:

bash

Copy code

cd path/to/your/project

1. **Initialize Git**:

bash

Copy code

git init

1. **Add Files**:

bash

Copy code

git add .

1. **Commit Changes**:

bash

Copy code

git commit -m "Initial commit"

1. **Add Remote Repository** (if you have one):

bash

Copy code

git remote add origin REMOTE\_URL

1. **Push to Remote**:

bash

Copy code

git push -u origin main

### ****Summary****

* **Exact Directory**: Run git init in the directory where your .py files are located to initialize a repository specifically for that project.
* **One Folder Up**: You can initialize Git in a parent directory if you want to track multiple subdirectories or projects within that folder.

By initializing Git in the appropriate directory, you ensure that all relevant files and changes are tracked as expected.

$ git add fatal: not a git repository (or any of the parent directories): .git

ChatGPT

The error message "fatal: not a git repository (or any of the parent directories): .git" indicates that the directory you're currently in is not part of a Git repository. This can happen for several reasons:

### ****Common Causes and Solutions****

#### ****1. Initialize Git Repository****

**Problem**: You haven't initialized a Git repository in the directory.

**Solution**: Navigate to your project directory and initialize a Git repository with the following command:

bash

Copy code

cd path/to/your/project

git init

This command creates a .git directory in your project folder, setting up Git to track changes.

#### ****2. Check for**** .git ****Directory****

**Problem**: You might be in the wrong directory, or the .git directory is missing or deleted.

**Solution**: Ensure you’re in the correct directory where your Git repository was initialized. Check for the .git directory:

bash

Copy code

ls -a

If the .git directory is missing, you need to reinitialize the repository by running git init in the correct directory.

#### ****3. Verify Git Installation****

**Problem**: Git might not be installed correctly or isn't recognized by your terminal.

**Solution**: Check if Git is installed and configured properly:

bash

Copy code

git --version

If Git is not installed, download and install it from the [official Git website](https://git-scm.com/).

#### ****4. Navigate to a Subdirectory****

**Problem**: You might be in a subdirectory of your project that doesn’t contain the .git directory.

**Solution**: Navigate to the root directory of your project where .git is located and try running git add again.

bash

Copy code

cd path/to/your/project

git add .

#### ****5. Re-clone the Repository****

**Problem**: The repository might be corrupted or missing essential files.

**Solution**: If the repository was cloned from a remote and it’s not functioning correctly, you might consider re-cloning it:

1. **Remove the existing repository (if necessary)**:

bash

Copy code

rm -rf path/to/your/project

1. **Clone the repository again**:

bash

Copy code

git clone REMOTE\_URL path/to/your/project

Replace REMOTE\_URL with your repository’s URL.

### ****Steps to Correct the Issue****

1. **Ensure You're in the Right Directory**:

Navigate to your project directory where the .git directory should be located.

bash

Copy code

cd path/to/your/project

1. **Initialize Git Repository**:

If .git is missing, initialize the repository:

bash

Copy code

git init

1. **Add Files**:

Once you’re in the right directory and have initialized Git, add your files:

bash

Copy code

git add .

1. **Commit Changes**:

After adding files, commit the changes:

bash

Copy code

git commit -m "Your commit message"

1. **Set Up Remote Repository**:

If you have a remote repository, add it:

bash

Copy code

git remote add origin REMOTE\_URL

1. **Push Changes**:

Push your changes to the remote repository:

bash

Copy code

git push -u origin main

### ****Summary****

The error generally means you're not in a Git repository directory. To resolve it:

* Ensure you're in the correct directory.
* Initialize Git if necessary.
* Check for the .git directory.
* Re-clone the repository if needed.

If you follow these steps, you should be able to resolve the issue and continue using Git effectively.

Top of Form

Bottom of Form