FACULDADE ENGENHARIA

Departamento de Informática

Bioinformática (15861, 13422)

Python Loops: Tips & Tricks

FACULDADE ENGENHARIA

Departamento de Informática

Loop: Analogy

FACULDADE ENGENHARIA

Departamento de Informática

Loop: Analogy

Consider a for loop as a music playlist. The playlist contains a set number of songs that you want to listen to in sequence.

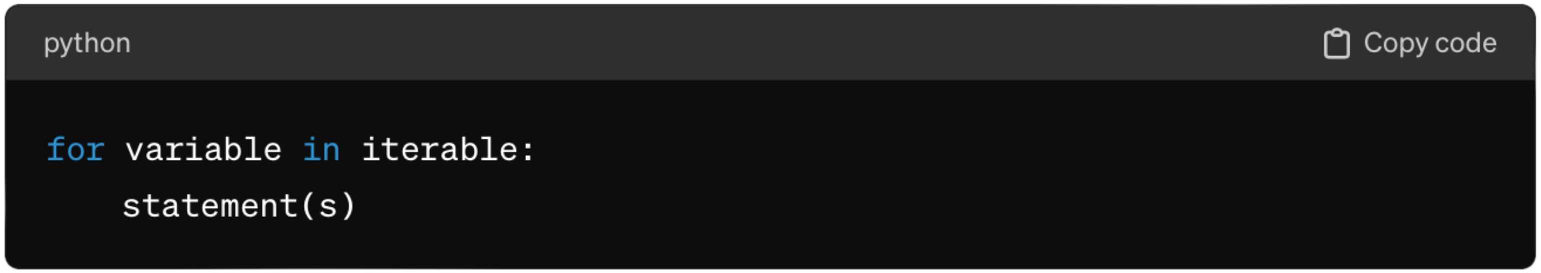
- **Initialization**: Creating the playlist is like setting up the for loop. You decide on the list of songs (the iterable) you want to play.
- **Condition**: Just as the music player checks if there are more songs to play, the for loop checks if there are more items in the sequence to iterate over.
- **Loop Body**: Playing a song from the playlist represents the body of the loop where the code is executed for each item in the sequence.
- **Update**: After a song finishes playing, the player automatically moves to the next song. Similarly, the loop variable moves to the next item in the sequence.
- **End Condition Check**: The player checks after each song if there are more songs to play, just like the for loop checks if it should continue with the next iteration.
- **End**: Once all songs have been played, the playlist ends. Similarly, when there are no more items to iterate, the for loop concludes.

In both the *for* loop and the playlist, the process is automatic; each item (or song) is handled in turn without the need for user intervention to proceed to the next one.

Loop

The “for” workflow is as follows:

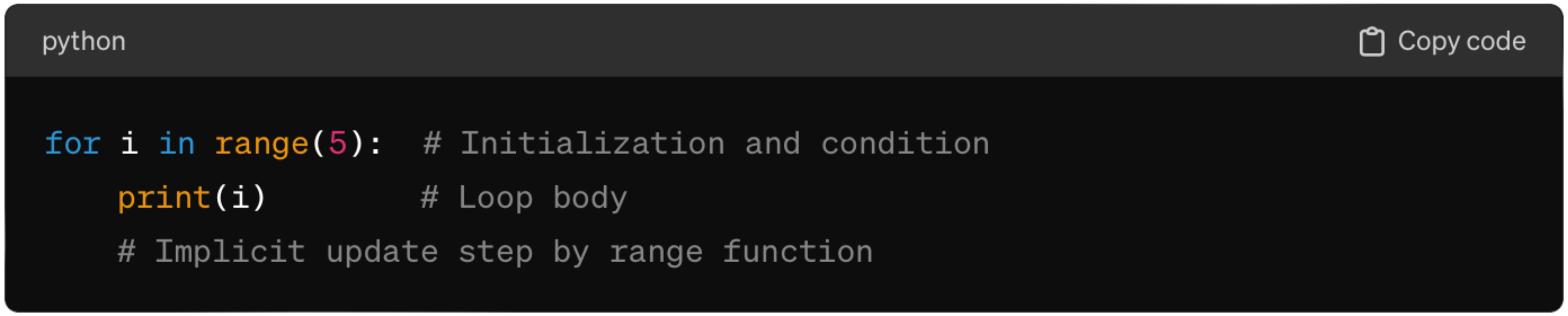
1. Start: The start of the loop.
2. Initialization: Define the loop variable and set the starting point.
3. Condition: Check the loop condition (typically, whether the loop variable meets a certain criterion to continue the loop).
4. Loop Body: The set of actions that are executed in each iteration of the loop.
5. Update: Modify the loop variable (increment or decrement) to progress the loop.
6. End Condition Check: Return to the condition step to check if the next iteration should occur.
7. End: Exit the loop once the condition is no longer met.



python Copy code

```python
for variable in iterable:
    statement(s)
```

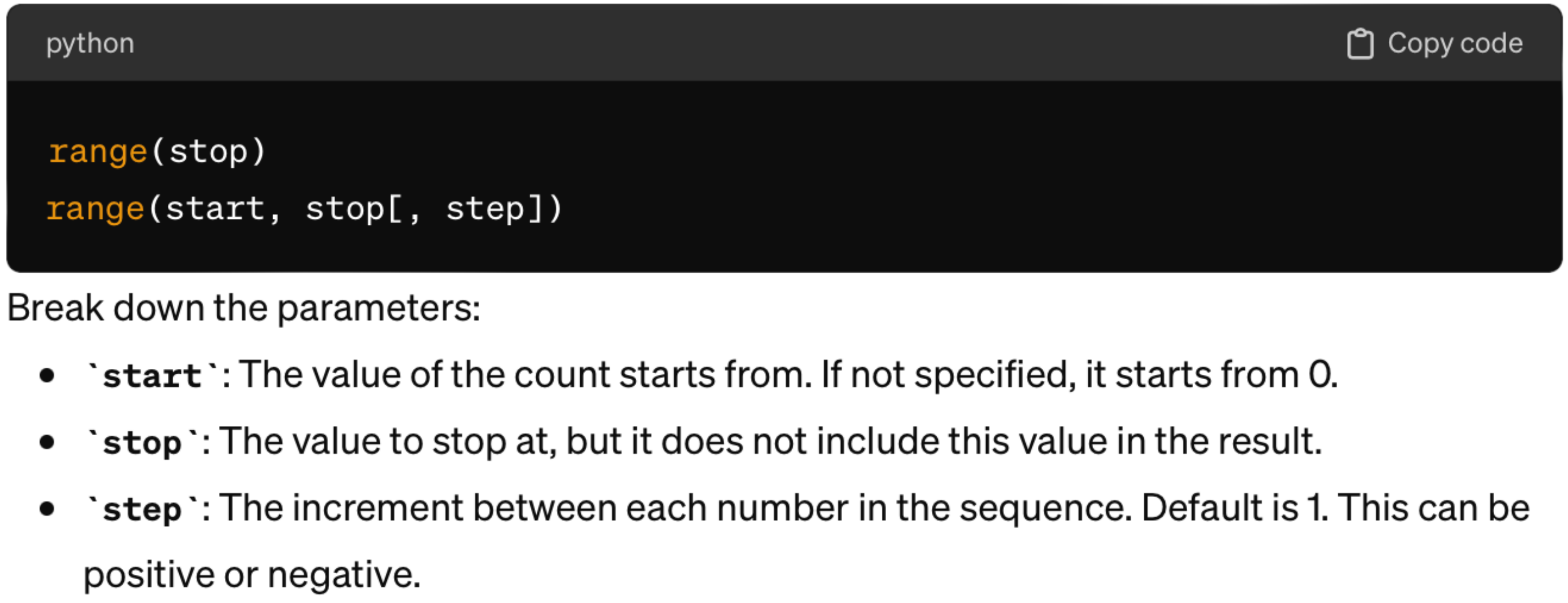
- Break down the components:
  - `variable` — a placeholder that takes the value of each item inside the iterable as the loop runs.
  - `iterable` — a collection of items over which the loop will run.
  - `statement(s)` — code that executes for each item in the iterable.



python Copy code

```python
for i in range(5):   # Initialization and condition
    print(i)         # Loop body
    # Implicit update step by range function
```

- In this case, i is initialized to 0, and the loop continues as long as i is less than 5, with i being incremented by 1 after each iteration.

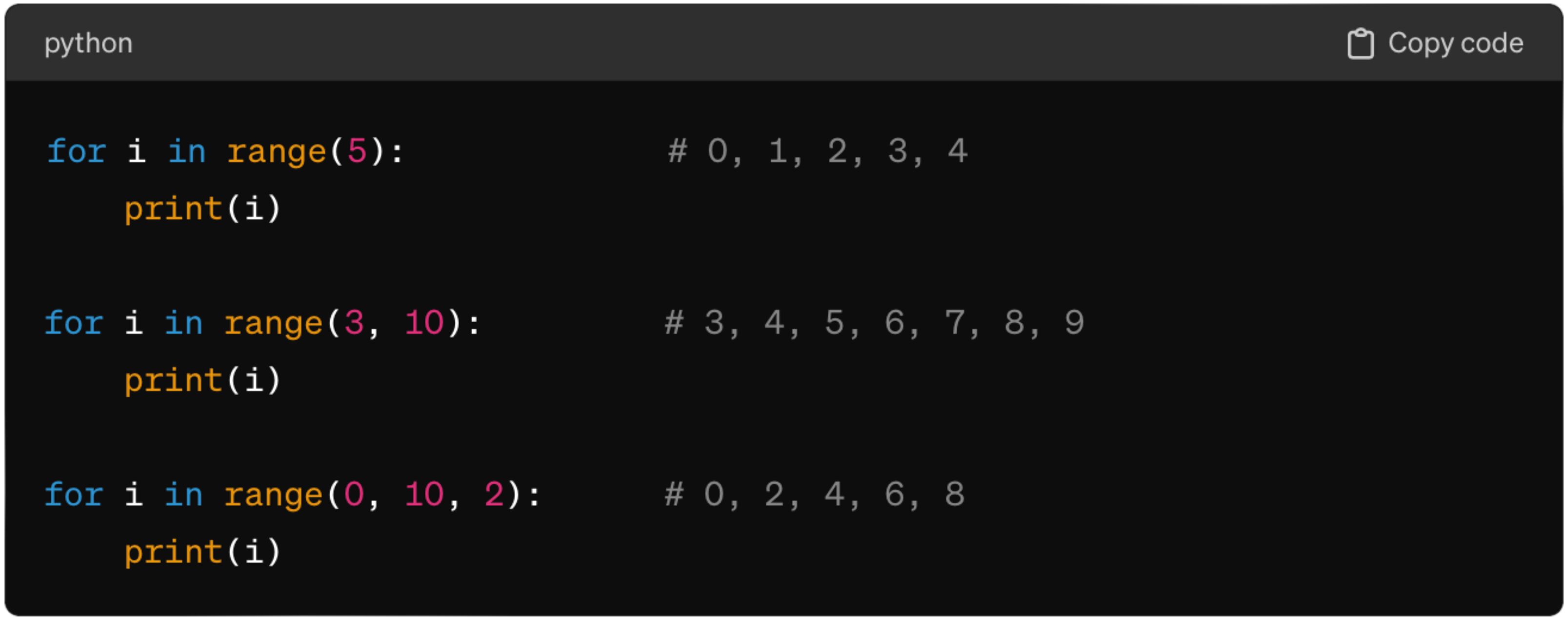


python Copy code

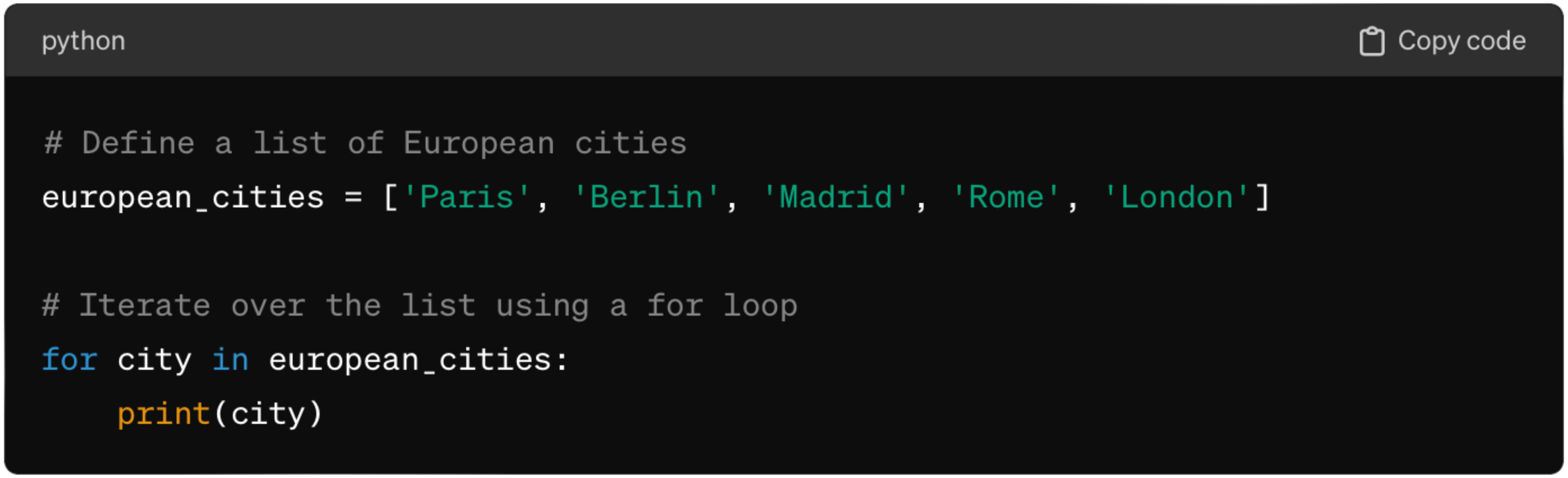
```python
range(stop)
range(start, stop[, step])
```

Break down the parameters:

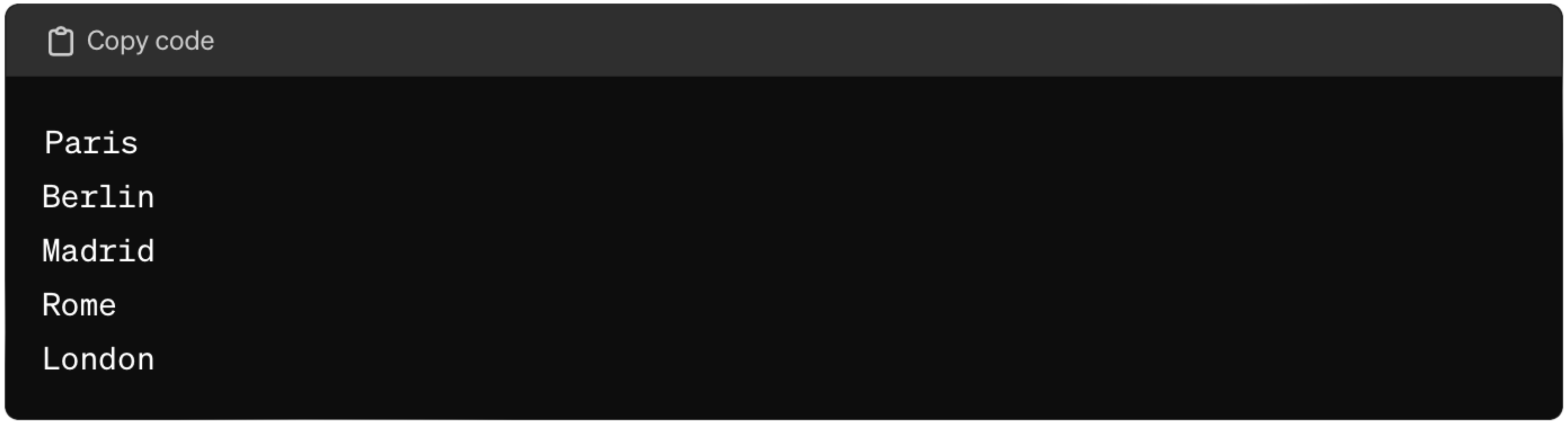
- `start`: The value of the count starts from. If not specified, it starts from 0.
- `stop`: The value to stop at, but it does not include this value in the result.
- `step`: The increment between each number in the sequence. Default is 1. This can be positive or negative.



```python
for i in range(5):             # 0, 1, 2, 3, 4
    print(i)

for i in range(3, 10):         # 3, 4, 5, 6, 7, 8, 9
    print(i)

for i in range(0, 10, 2):      # 0, 2, 4, 6, 8
    print(i)
```



```python
# Define a list of European cities
european_cities = ['Paris', 'Berlin', 'Madrid', 'Rome', 'London']

# Iterate over the list using a for loop
for city in european_cities:
    print(city)
```



```
Paris
Berlin
Madrid
Rome
London
```

HOW
WHY
WHEN
WHERE
WHY
WHERE
WHERE
WHO
WHEN
WHAT
WHAT
WHY
WHAT
WHEN
HOW
WHAT
WHO
WHEN
WHERE
WHY
HOW