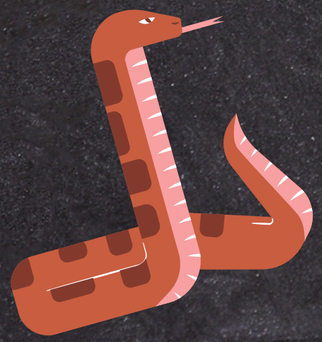
M. Amin HosseinNiya | Ali Rashedi

Python Programming

A brief of the content!



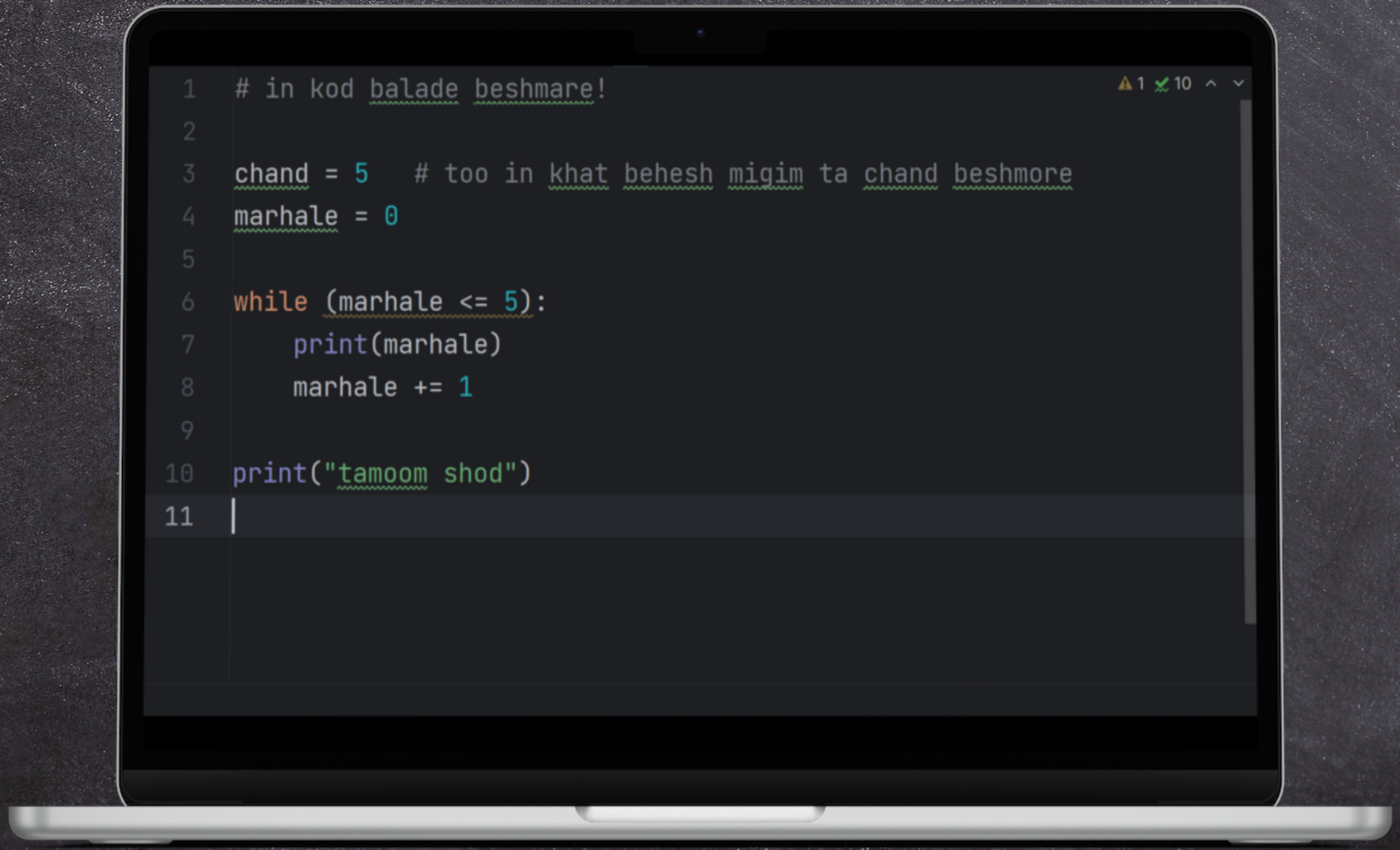
Session 02

M. Amin HosseinNiya | Ali Rashedi

While loop

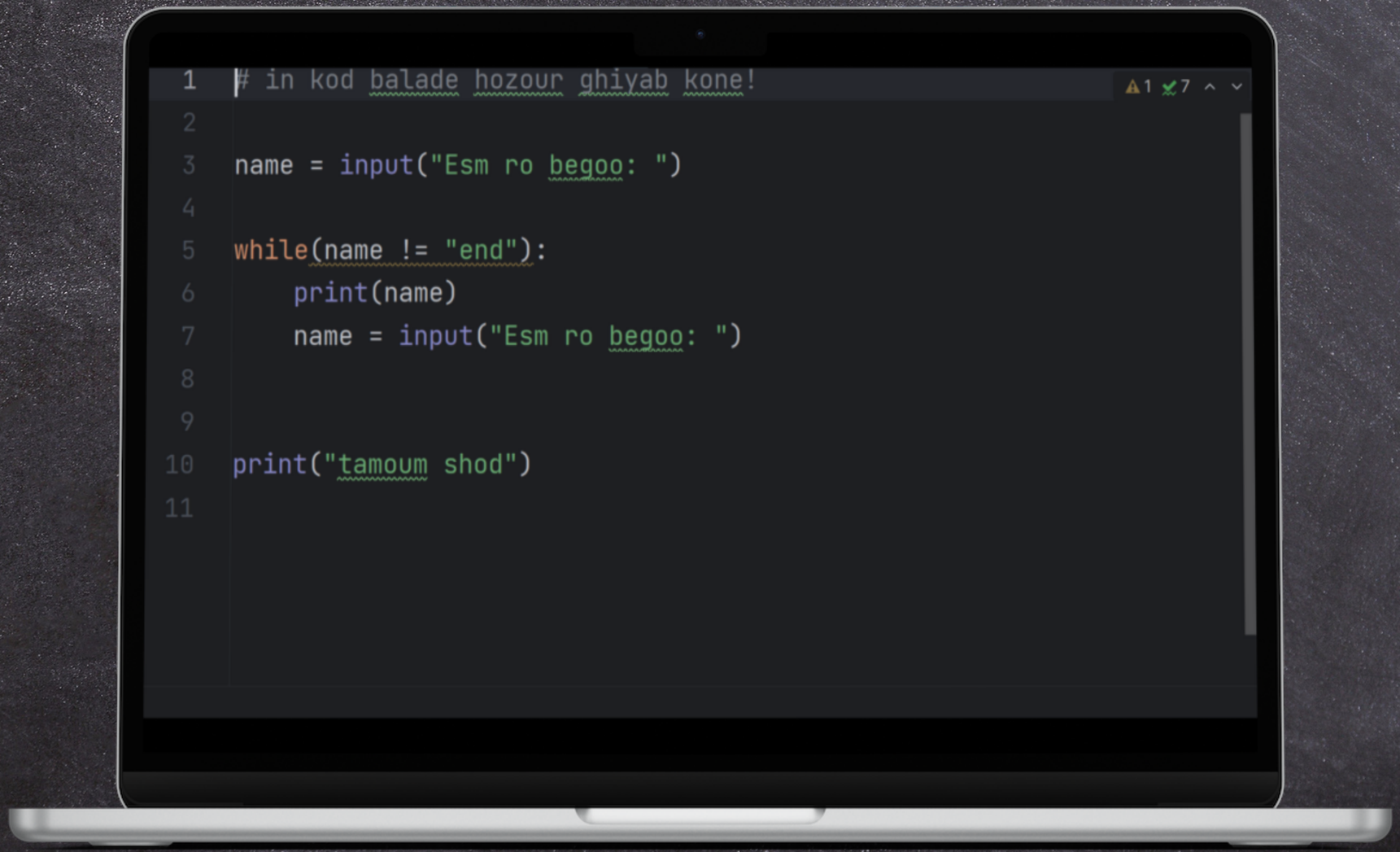
A 'While' loop is a type of loop in Python that allows you to repeat a block of code while a certain condition is true.

While loop in action



While_example1_Beshmar.py file

While loop in action

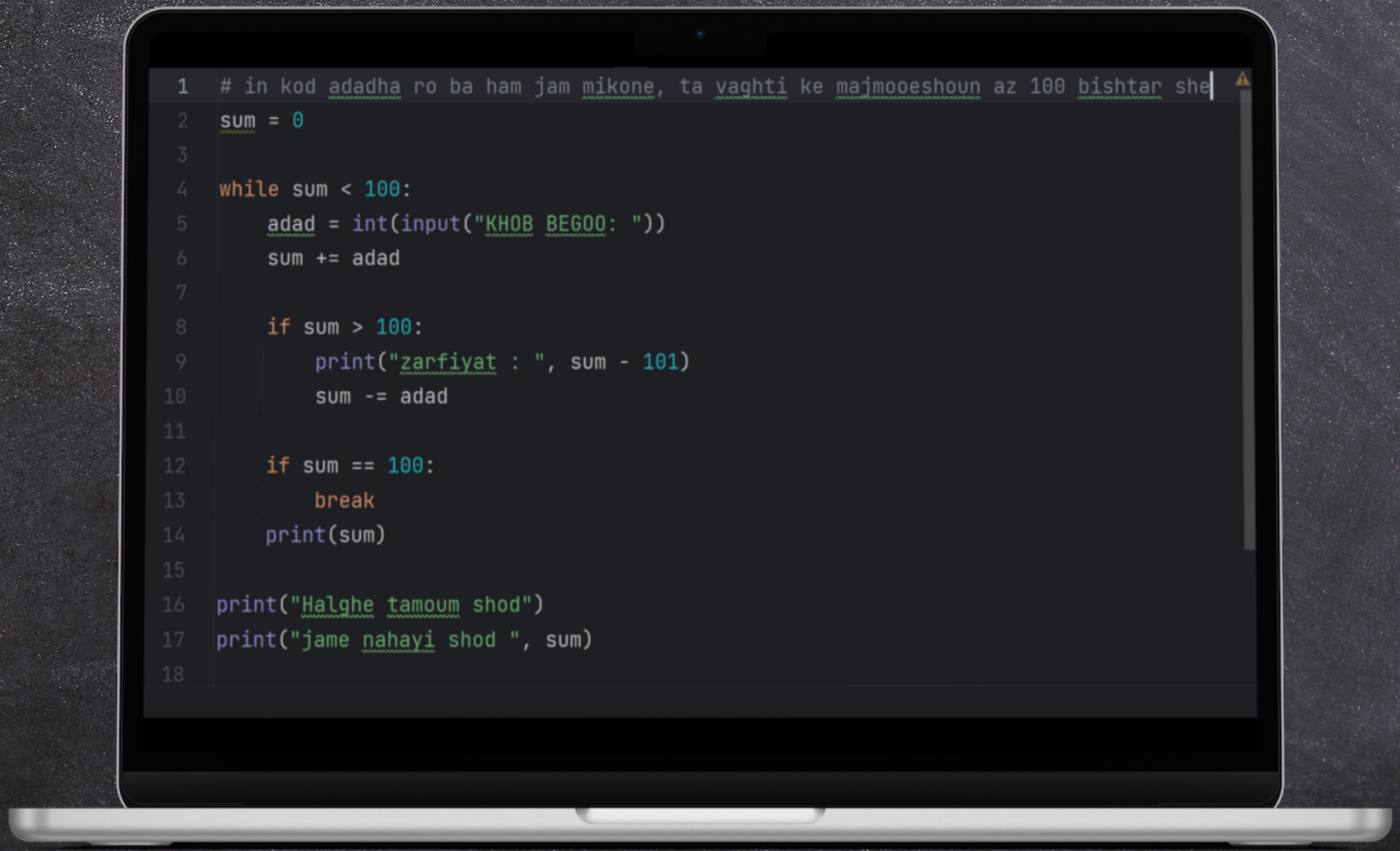


While_example2_HozourGhiyab.py file

Break statement!

A 'break' statement is a way to exit a loop prematurely, even if the loop condition is still true.

Break statement in action



```python
# in kod adadha ro ba ham jam mikone, ta vaghti ke majmooeshoun az 100 bishtar she
sum = 0

while sum < 100:
    adad = int(input("KHOB BEGOO: "))
    sum += adad

    if sum > 100:
        print("zarfiyat : ", sum - 101)
        sum -= adad

    if sum == 100:
        break
    print(sum)

print("Halghe tamoum shod")
print("jame nahayi shod ", sum)
```

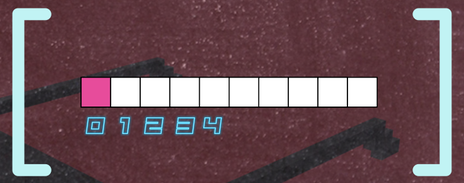
While_example3_JamZadan.py

M. Amin HosseinNiya | Ali Rashedi

Liiiiist

Lists are one of the fundamental data structures in Python. They are used to store and organize collections of items. Here are some key characteristics of lists:

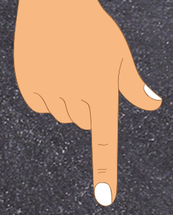
- Ordered: Lists maintain the order of elements as they are added.
- Mutable: Elements in a list can be modified, added, or removed.
- Heterogeneous: Lists can contain different data types, such as numbers, strings, or even other lists.



python

```
fruits = ["apple", "banana", "orange", "grape"]
```

In this example, we have a list called "fruits" that contains four elements. The order of the elements is preserved, and we can access each element by its position in the list.



Indexing in list

In Python, elements in a list can be accessed using indexes. The index represents the position of an element in the list. It's important to note that indexing in Python starts from 0. Here's an example to illustrate list indexing:

NEXT ➡

```python
fruits = ["apple", "banana", "orange", "grape"]

# Accessing elements using positive indexes
print(fruits[0])   # Output: "apple"
print(fruits[2])   # Output: "orange"

# Accessing elements using negative indexes
print(fruits[-1])  # Output: "grape"
print(fruits[-3])  # Output: "banana"
```

In this example, we have a list called "fruits." By specifying the index in square brackets, we can access individual elements in the list. Positive indexes count from the beginning of the list, while negative indexes count from the end.

List indexing allows us to retrieve and manipulate specific elements within a list, enabling us to perform various operations and computations on our data.

Basic List Operations

append():

- Syntax: list.append(element)
- Adds an element to the end of the list.
- Example:

```python
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)  # Output: [1, 2, 3, 4]
```

pop():

- Syntax: list.pop()
- Removes and returns the last element from the list.
- Example:

```python
numbers = [1, 2, 3, 4]
last_number = numbers.pop()
print(last_number)  # Output: 4
print(numbers)      # Output: [1, 2, 3]
```

Basic List Operations

remove():

- Syntax: list.remove(element)
- Removes the first occurrence of the specified element from the list.
- Example:

```python
fruits = ["apple", "banana", "orange", "banana"]
fruits.remove("banana")
print(fruits)  # Output: ["apple", "orange", "banana"]
```

sort():

- Syntax: list.sort()
- Sorts the elements of the list in ascending order.
- Example:

```python
numbers = [4, 2, 1, 3]
numbers.sort()
print(numbers)  # Output: [1, 2, 3, 4]
```

Basic List Operations

len():

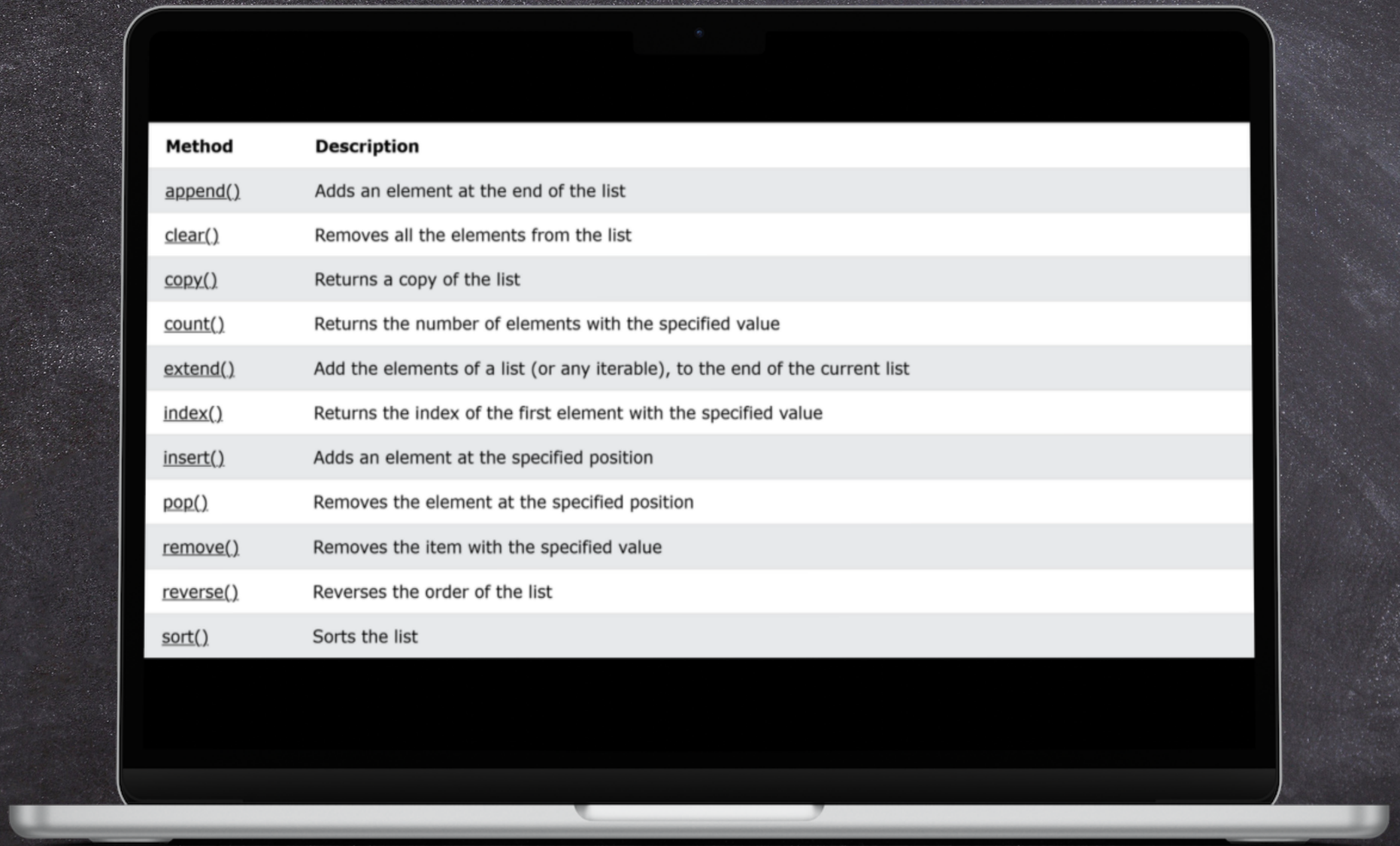
- Syntax: len(list)
- Returns the number of elements in the list.
- Example:

```python
fruits = ["apple", "banana", "orange"]
length = len(fruits)
print(length)  # Output: 3
```

sum():

- Syntax: sum(list)
- Calculates the sum of all the elements in the list (if they are numeric).
- Example:

```python
numbers = [1, 2, 3, 4]
total = sum(numbers)
print(total)  # Output: 10
```



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

M. Amin HosseinNiya | Ali Rashedi

Fooooooor

For loops in Python are a powerful construct that allows us to iterate over a sequence of elements. They are used when we want to perform a set of actions repeatedly for each item in the sequence. The general syntax of a for loop in Python is as follows:

```python
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    print(fruit)
```

Break and Continue

Python provides two useful statements, "break" and "continue", to control the flow of execution within loops.

The "break" statement is used to prematurely exit the loop when a certain condition is met. It completely terminates the loop and continues execution with the next statement outside of the loop.

On the other hand, the "continue" statement is used to skip the rest of the current iteration and move on to the next iteration of the loop.

python

```python
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    if fruit == "banana":
        break
    print(fruit)
```

python

```python
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    if fruit == "banana":
        continue
    print(fruit)
```

Presented by

Ali Rashedi

alirashedi2014@gmail.com

ali-rashedi

M. Amin HosseinNiya

m.amin.hosseinniya@gmail.com

mohammadamin-hosseinniya