ENSF 381
Full Stack Web Development

Lecture 27: Collections and Functions

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Arrays/Lists.
- Dictionary.
- Functions.

What are Arrays?

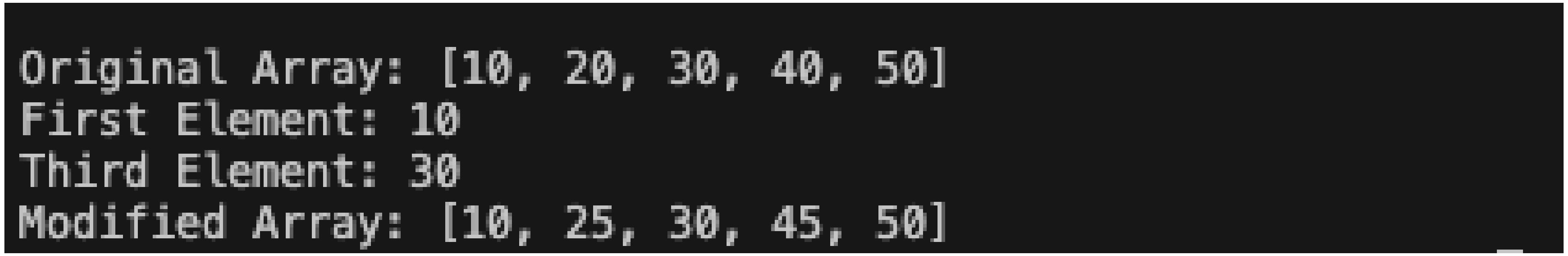
- Arrays are data structures that store multiple items in a single variable.
- In Python, arrays are represented using lists.
- Accessing elements using index (indexing starts from 0).
- Syntax:

```
my_array = [element1, element2, element3]
```

Arrays - example

```
# Define an array (list)
my_array = [10, 20, 30, 40, 50]

# Accessing elements
first_element = my_array[0]
third_element = my_array[2]

print("Original Array:", my_array)
print("First Element:", first_element)
print("Third Element:", third_element)

# Modifying elements
my_array[1] = 25  # Update the second element
my_array[3] += 5  # Increment the fourth element by 5

print("Modified Array:", my_array)
```

Arrays - example



```
Original Array: [10, 20, 30, 40, 50]
First Element: 10
Third Element: 30
Modified Array: [10, 25, 30, 45, 50]
```

Arrays - methods

- `len`: is a built-in function used to get the length (the number of elements) of a sequence, such as a list, tuple, or string.

Arrays - methods

Code snippet

```
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

Output

$\longrightarrow$

```
5
```

Arrays - methods

- `len`: is a built-in function used to get the length (the number of elements) of a sequence, such as a list, tuple, or string.

- `append`: adds a specified element to the end of the list.

Arrays - methods

Code snippet

```
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

Output

```
5
```

```
my_list = [1, 2, 3, 4, 5]
my_list.append(6)
print(my_list)
```

```
[1, 2, 3, 4, 5, 6]
```

Arrays - methods

- `len`: is a built-in function used to get the length (the number of elements) of a sequence, such as a list, tuple, or string.

- `append`: adds a specified element to the end of the list.

- `insert`: inserts a specified element at a given position (index) in the list.

Arrays - methods

Code snippet

Output

```
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

```
5
```

```
my_list = [1, 2, 3, 4, 5]
my_list.append(6)
print(my_list)
```

```
[1, 2, 3, 4, 5, 6]
```

```
my_list = [1, 2, 3, 4, 5]
my_list.insert(2, 7)
print(my_list)
```

```
[1, 2, 7, 3, 4, 5]
```

Arrays - methods

- `len`: is a built-in function used to get the length (the number of elements) of a sequence, such as a list, tuple, or string.
- `append`: adds a specified element to the end of the list.
- `insert`: inserts a specified element at a given position (index) in the list.
- `remove`: removes the first occurrence of a specified value from the list.

Arrays - methods

Code snippet

Output

```
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

```
5
```

```
my_list = [1, 2, 3, 4, 5]
my_list.append(6)
print(my_list)
```

```
[1, 2, 3, 4, 5, 6]
```

```
my_list = [1, 2, 3, 4, 5]
my_list.insert(2, 7)
print(my_list)
```

```
[1, 2, 7, 3, 4, 5]
```

```
my_list = [1, 2, 3, 4, 5]
my_list.remove(4)
print(my_list)
```

```
[1, 2, 3, 5]
```

Arrays - methods

- `len`: is a built-in function used to get the length (the number of elements) of a sequence, such as a list, tuple, or string.
- `append`: adds a specified element to the end of the list.
- `insert`: inserts a specified element at a given position (index) in the list.
- `remove`: removes the first occurrence of a specified value from the list.
- `extend`: appends the elements of an iterable (e.g., list, tuple) to the end of the current list, effectively extending its length.

Arrays - methods

Code snippet

Output

```
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

```
5
```

```
my_list = [1, 2, 3, 4, 5]
my_list.append(6)
print(my_list)
```

```
[1, 2, 3, 4, 5, 6]
```

```
my_list = [1, 2, 3, 4, 5]
my_list.insert(2, 7)
print(my_list)
```

```
[1, 2, 7, 3, 4, 5]
```

```
my_list = [1, 2, 3, 4, 5]
my_list.remove(4)
print(my_list)
```

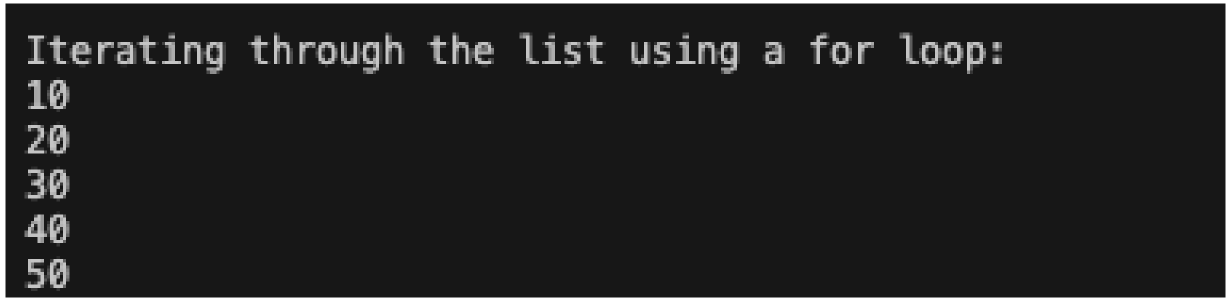
```
[1, 2, 3, 5]
```

```
my_list = [1, 2, 3, 4, 5]
additional_elements = [6, 7, 8]
my_list.extend(additional_elements)
print(my_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

Iterate through list's elements

```python
# Define a list
my_list = [10, 20, 30, 40, 50]

# Using a for loop to iterate through the list
print("Iterating through the list using a for loop:")
for element in my_list:
    print(element)
```



```
Iterating through the list using a for loop:
10
20
30
40
50
```

Dictionary

- Dictionary is versatile and powerful data structures in Python.

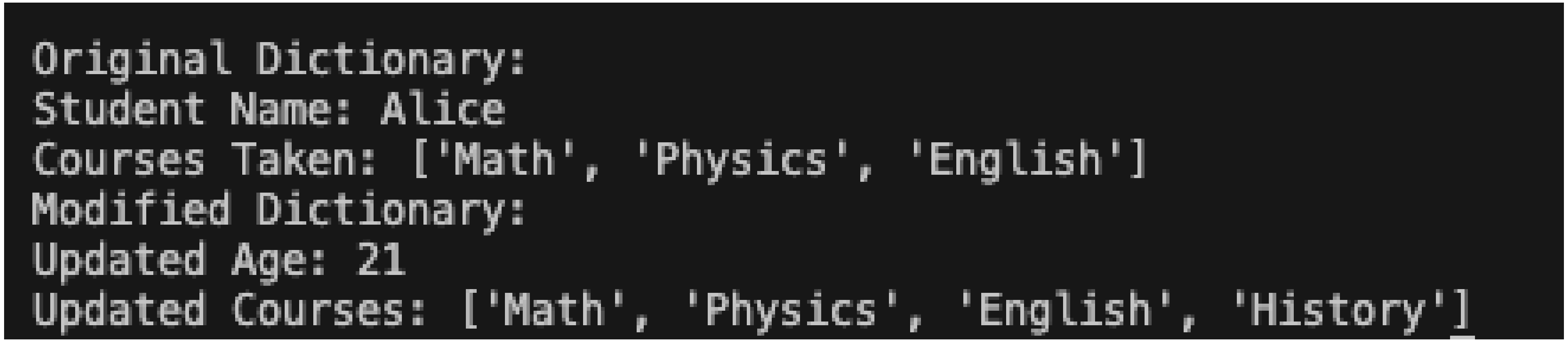
- They are unordered collections of key-value pairs, providing fast and efficient data retrieval.

- Use Cases:
  - Ideal for scenarios where data retrieval based on a unique identifier (key) is crucial.
  - Used to represent real-world entities and their associated attributes.

Dictionary - example

```python
# Define a dictionary
student_info = {
    'name': 'Alice',
    'age': 20,
    'grade': 'A+',
    'courses': ['Math', 'Physics', 'English']
}

# Accessing elements
student_name = student_info['name']
courses_taken = student_info['courses']

print("Original Dictionary:")
print("Student Name:", student_name)
print("Courses Taken:", courses_taken)

# Modifying elements
student_info['age'] = 21  # Update the age
student_info['courses'].append('History')  # Add a new course

print("Modified Dictionary:")
print("Updated Age:", student_info['age'])
print("Updated Courses:", student_info['courses'])
```

Dictionary - example

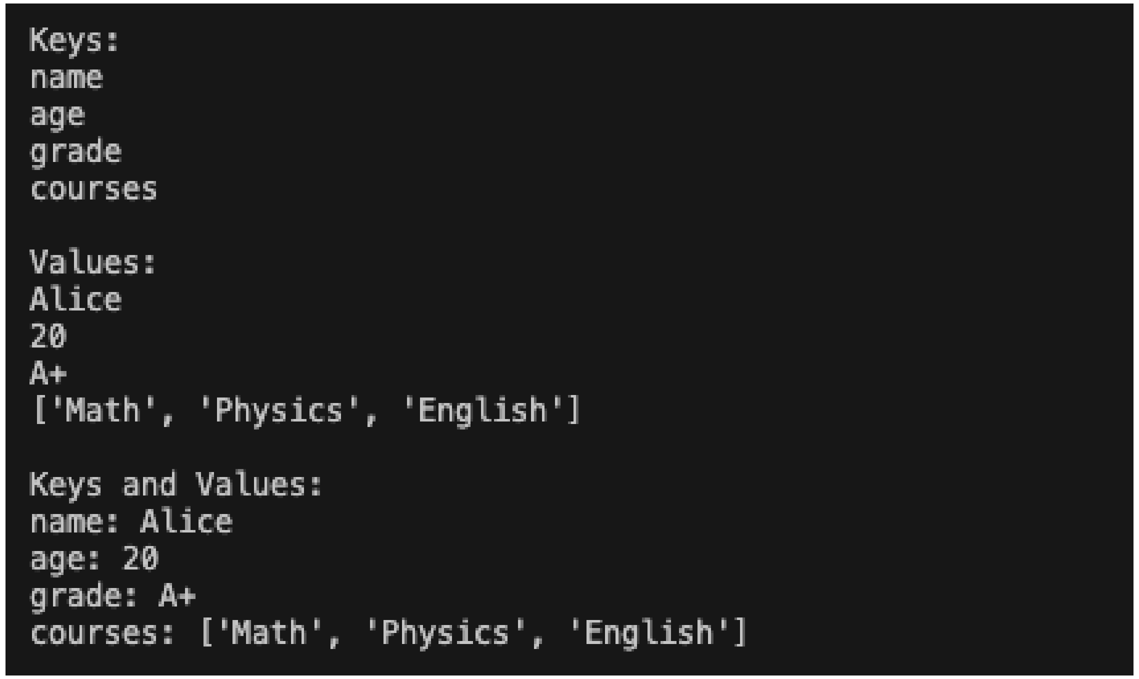


```
Original Dictionary:
Student Name: Alice
Courses Taken: ['Math', 'Physics', 'English']
Modified Dictionary:
Updated Age: 21
Updated Courses: ['Math', 'Physics', 'English', 'History']
```

Iterate through a dictionary's keys and values

```python
student_info = {
    'name': 'Alice',
    'age': 20,
    'grade': 'A+',
    'courses': ['Math', 'Physics', 'English']
}

# Print keys
print("Keys:")
for key in student_info.keys():
    print(key)
print() # Add an empty line to improve the readability of the output
# Print values
print("Values:")
for value in student_info.values():
    print(value)
print()
# Print both keys and values
print("Keys and Values:")
for key, value in student_info.items():
    print(f"{key}: {value}")
```

Iterate through a dictionary's keys and values



```
Keys:
name
age
grade
courses

Values:
Alice
20
A+
['Math', 'Physics', 'English']

Keys and Values:
name: Alice
age: 20
grade: A+
courses: ['Math', 'Physics', 'English']
```

Functions

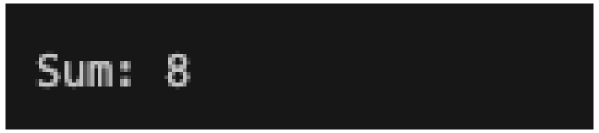
- A function in Python is a reusable and self-contained block of code designed to perform a specific task.
- Functions help **modularize code**, making it more organized, readable, and easier to maintain.
- Define a function using the `def` keyword followed by the function name and parameters.
- Use indentation to define the block of code within the function.

Function - syntax

```
def function_name(parameter1, parameter2, ...):
    """
    Docstring: Description of the function; provide information about the function
    Optional multiline documentation.
    """
     # Function body (code block)
     # ...
     # Optional return statement
     return result
```

Functions - example

```python
def add_numbers(x, y):
    """Add two numbers and return the result."""
    result = x + y
    return result

# Calling the function
sum_result = add_numbers(3, 5)
print("Sum:", sum_result)
```



```
Sum: 8
```

Question...

```
def modify_variable(variable_to_modify):
  # Modify the variable
  variable_to_modify =  variable_to_modify * 2

score = 30
print(score)

modify_variable(score)
print(score)
```

Output:

```
30
30
```

Default parameter value

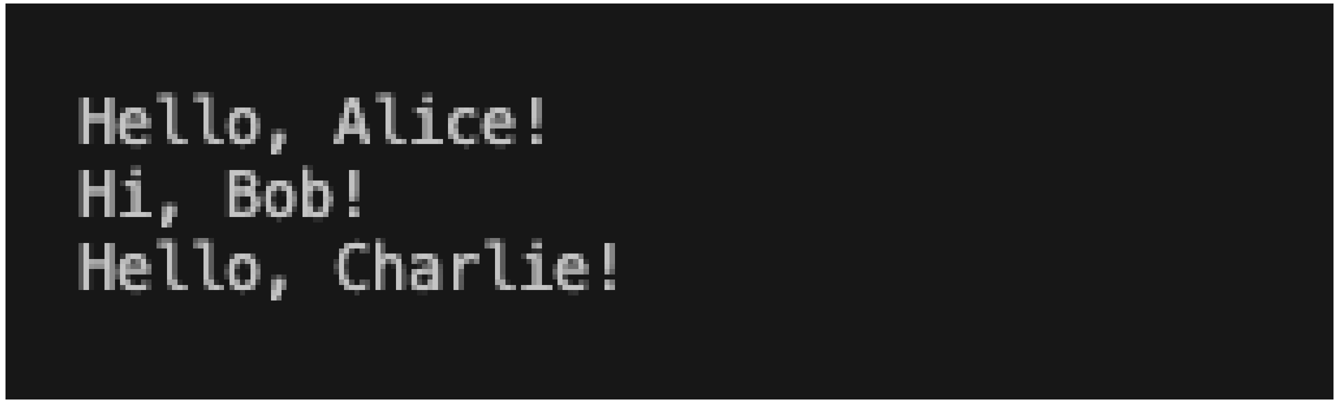
```python
def greet(name, greeting="Hello"):
  print(f"{greeting}, {name}!")
```

```python
# Using the default greeting
greet("Alice")
```

```python
# Providing a custom greeting
greet("Bob", "Hi")
```

```python
# Using the default greeting with a different name
greet("Charlie")
```

Default parameter value



```
Hello, Alice!
Hi, Bob!
Hello, Charlie!
```

Calculate the average of students' grades

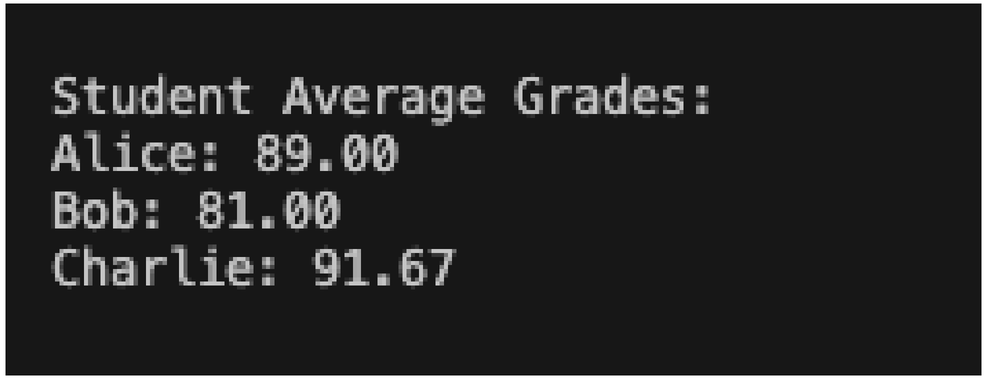
```
students = [
  {'name': 'Alice', 'grades': [85, 90, 92]},
  {'name': 'Bob', 'grades': [78, 85, 80]},
  {'name': 'Charlie', 'grades': [92, 88, 95]}]
```

```
def calculate_average(grades): # Function to calculate the average grade for a student.
  if len(grades) > 0:
      total = sum(grades) # A built-in function that calculates the sum of elements in a list
      return total / len(grades)
  else:
      return 0
```

```
average_grades = {} # Dictionary to store average grades.
for student in students: # Calculate and store average grades for each student.
  name = student['name']
  grades = student['grades']
  average = calculate_average(grades)
  average_grades[name] = average
```

```
print("Student Average Grades:")
for name, average in average_grades.items():
  print(f"{name}: {average:.2f}")
```

Calculate the average of students' grades



```
Student Average Grades:
Alice: 89.00
Bob: 81.00
Charlie: 91.67
```

Questions

?