Part-4

Lists in Python

Description

- Lists in Python are ordered collections of objects, values, or items of different types
- Lists are defined by enclosing elements in square brackets and separating them with commas
- Key characteristics of lists:
- Ordered: Elements have a defined order that does not change when new items are added
- Indexed: Elements can be accessed by index values
- Heterogeneous: Can store different data types (numbers, strings, other lists, etc.)
- Mutable: Elements can be changed, added, or removed after creation
- Allow duplicates: Lists can have items with the same value

Important List Methods

- **append()**: Adds an element at the end of the list
- **insert()**: Adds an element at the specified position
- **extend()**: Adds the elements of a list to the end of the current list
- **index()**: Returns the index of the first element with the specified value
- **remove()**: Removes the item with the specified value
- **sort()**: Sorts the list
- **reverse()**: Reverses the order of the list

Accessing List Elements

- Lists are indexed starting from 0
- Negative indices can be used to access elements from the end of the list
- Slicing can be used to access a range of elements

Slicing Lists

- Syntax: `list[start:stop]`
- Extracts elements from the start index (inclusive) to the stop index (exclusive)
- Can also use a single index to access an element at that position

Tuples in Python

Description

- Tuples are ordered collections of objects, values, or items of different types
- Tuples are defined by enclosing elements in parentheses and separating them with commas
- Key characteristics of tuples:
- Ordered: Elements have a defined order that does not change
- Indexed: Elements can be accessed by index values
- Heterogeneous: Can store different data types
- Immutable: Elements cannot be changed, added, or removed after creation
- Allow duplicates: Tuples can have items with the same value

Tuple Methods

- **index()**: Searches the tuple for a specified value and returns the position
- **count()**: Returns the number of times a specified value occurs in the tuple

Accessing Tuple Elements

- Tuples are indexed starting from 0
- Negative indices can be used to access elements from the end of the tuple
- Slicing can be used to access a range of elements
- Tuples support concatenation and repetition operations

Dictionaries in Python

Description

- Dictionaries in Python are collections of key-value pairs
- Dictionaries are defined by enclosing key-value pairs in curly braces and separating them with commas
- Key characteristics of dictionaries:
  - Ordered (from Python 3.7 onwards)
  - Elements cannot be accessed by index
  - Can store different data types
  - Mutable: Key-value pairs can be changed, added, or removed
  - No duplicate keys allowed (but values can be duplicated)

Dictionary Methods

- **clear()**: Removes all elements from the dictionary
- **get()**: Returns the value of the specified key
- **keys()**: Returns a list of all the keys in the dictionary
- **values()**: Returns a list of all the values in the dictionary
- **update()**: Updates the dictionary with the specified key-value pairs
- **pop()**: Removes the element with the specified key
- **popitem()**: Removes the last inserted key-value pair

Accessing Dictionary Elements

- Dictionaries use keys to access their values
- Values can be accessed using the key in square brackets or the `get()`method

Looping through Dictionaries

- Can use a `for` loop to iterate through the keys or key-value pairs in a dictionary

Conditional Statements in Python

Description

- Conditional statements in Python allow you to execute different blocks of code based on certain conditions
- The `if`, `elif`, and `else` statements are used to implement conditional logic

`if` Statement

- The `if` statement checks a condition and executes the code block if the condition is true
- Syntax:

```
if condition:
{
        if statement
}
```

`if-elif-else` Statements

- The `elif` statement checks additional conditions if the previous `if` or `elif` conditions were false
- The `else` statement executes a default code block if all previous conditions were false
- Syntax:

```
if condition1:    # code blockelif condition2:    # code blockelse:    # code block
```

Example: Shipping Cost Calculation

- Calculates the shipping cost for a package based on the total amount and the destination state
- Uses nested `if-elif-else` statements to determine the shipping cost
- Handles invalid state names by printing an "invalid state" message

While Loops in Python

Description

- While loops in Python execute a block of code as long as a certain condition is true
- The condition is evaluated before each iteration of the loop
- The loop continues until the condition becomes false or a `break` statement is encountered

Syntax

```
while condition:    # code block
```

Example 1: Simple While Loop

- Initializes a variable `i` to 0
- Increments `i` in each iteration of the loop
- Prints the value of `i` with a hash symbol after it
- Continues until `i` is less than 10

Example 2: Condition-Controlled Loop

- Initializes a variable `count` to 0
- Checks if `count` is less than or equal to 5
- Prints the message "the condition is true" and increments `count`
- Continues until `count` is no longer less than or equal to 5

Example 3: Using `break` Statement

- Sets the condition of the while loop to `True`(an infinite loop)
- Prompts the user to enter their name
- Uses the `break` statement to exit the loop when the user enters a name

Table: List Methods

Method	Description
`append()`	Adds an element at the end of the list
`insert()`	Adds an element at the specified position
`extend()`	Adds the elements of a list to the end of the current list
`index()`	Returns the index of the first element with the specified value
`remove()`	Removes the item with the specified value
`sort()`	Sorts the list
`reverse()`	Reverses the order of the list