SANJANA DEVI

AI/ML Student | Intern @ CodroidHub Private Limited | First-Year Engineering Student.

Artificial Intelligence / Machine Learning Bootcamp

Day 5: Data Structures

Data Structures in Python

AGENDA

1. **Introduction to Data Structures**
2. **Lists**
3. **Tuples**
4. **Sets**
5. **Strings**
6. **Dictionaries**

Learning Outcomes

- Ability to manipulate data using lists, tuples, sets, and dictionaries.
- Knowledge of when and why to use each data structure.
- Understanding how to perform basic operations on each structure (adding, removing, accessing elements).
- Ability to implement stacks and queues using Python data structures.
- Proficiency in solving common programming problems using the appropriate data structure.

1. Introduction to Data Structures in Python

What Are Data Structures?

Data structures are ways of organizing and storing data so it can be accessed and worked with efficiently. In Python, data structures are ways to store and organize data so you can use it efficiently. Python gives us some built-in data structures—you don't need to create them from scratch("means starting from the beginning — without using anything pre-made or already existing").

Types of Data Structures in Python:

- Lists
- Tuples
- Dictionaries
- Sets

In this notebook, we'll explore each one, step by step!

2. Lists in Python

A list is like a container or basket that holds items in order. You can change the items (it's mutable), and the items can be of any type.

How to Create a List:

Lists are defined using square brackets [].

Each item in a list is separated by a comma.

In [5]:
```
fruits = ["apple", "banana", "cherry"] # It is a list
```

Opreations in Lists

Data Structure	Function / Method	What It Does	Example
List	`append(x)`	Adds an item at the end	`fruits.append("mango")`
	`insert(i, x)`	Inserts item at position `i`	`fruits.insert(1, "orange")`
	`remove(x)`	Removes the first occurrence of `x`	`fruits.remove("apple")`
	`pop([i])`	Removes item at index `i` (last if not given)	`fruits.pop()`
	`sort()`	Sorts the list in order	`fruits.sort()`
	`reverse()`	Reverses the list order	`fruits.reverse()`
	`index(x)`	Finds index of first occurrence of `x`	`fruits.index("banana")`
	`count(x)`	Counts how many times `x` appears	`fruits.count("apple")`

In [9]:
```
# List Operations Example
fruits.append("orange")   # Adding an item
print(fruits)

fruits.remove("banana")   # Removing an item
print(fruits)

last_item = fruits.pop()  # Removing the last item
print("Last item removed:", last_item)
print(fruits)

# Slicing
print(fruits[0:2])  # Output: ['apple', 'cherry']
```

```
['apple', 'banana', 'cherry', 'orange']
['apple', 'cherry', 'orange']
Last item removed: orange
['apple', 'cherry']
['apple', 'cherry']
```

3. Tuples

A tuple is like a list, but once you create it, you can’t change it (it's immutable). Useful when your data should stay the same.

How to Create a Tuple:

- Tuples are defined using parentheses `()` .

In [12…
```
colors = ("red", "green", "blue") # it is a tuple
# Creating a tuple of dimensions
dimensions = (1920, 1080)
print("Width:", dimensions[0])
print("Height:", dimensions[1])
```

```
Width: 1920
Height: 1080
```

Operations in Tuples

| Tuple === 1.count(x) | Counts how many times x appears | colors.count("red") |

2. index(x)| Finds index of first occurrence of x | colors.index("blue") |

In [19…
```
##  Example
colors = ("red", "green", "blue")
r=colors.count("red")
b=colors.index("blue")
print("the count of red is ",r)
print("the index of blue is ",b)
```

```
the count of red is  1
the index of blue is  2
```

4. Sets

A set is an unordered collection of **unique items**. Unlike lists or tuples, sets do not allow duplicate elements. They are useful when you want to store items that should not repeat and when you need to perform operations like unions and intersections.

How to Create a Set:

- Sets are created using curly braces `{}` or the `set()` function.

Opreations in Sets

Operation Type	Function / Method	**Description **	Example
Create a Set	`set()` or `{}`	Creates a set with unique items	`s = {1, 2, 3}`
			`s = set([1, 2, 2, 3])` → `{1, 2, 3}`
Add Item	`add(x)`	Adds an item to the set	`s.add(4)`
Update Set	`update([a, b])`	Adds multiple items to the set	`s.update([5, 6])`
Remove Item	`remove(x)`	Removes `x` from the set; gives error if not found	`s.remove(2)`
Safe Remove	`discard(x)`	Removes `x` , but **no error** if not found	`s.discard(10)`
Remove Last Item	`pop()`	Removes a random item	`s.pop()`
Clear All	`clear()`	Empties the set	`s.clear()`

Operation	Method	Description	Example
Union	`set1.union(set2)`	Combines both sets (removes duplicates)	`a.union(b)`
	`set1	set2`	Same as above using symbol	`{1, 2}	{2, 3}` → `{1, 2, 3}`
Intersection	`set1.intersection(set2)`	Common elements in both sets	`a.intersection(b)`
	`set1 & set2`	Same as above using symbol	`{1, 2} & {2, 3}` → `{2}`
Difference	`set1.difference(set2)`	Items in set1 but **not** in set2	`{1, 2, 3}.difference({2})` → `{1, 3}`
	`set1 - set2`	Same as above using symbol	`{1, 2} - {2}` → `{1}`
Symmetric Difference	`set1.symmetric_difference(set2)`	Items **not common** in both sets	`{1, 2} ^ {2, 3}` → `{1, 3}`
	`set1 ^ set2`	Same as above using symbol	

In [22…
```
#Creating a sets
s = {1, 2, 3}
s = set([1, 2, 2, 3])# output {1, 2, 3}
```

5. Strings

String as a data structure too—because even though it stores characters and not collections like lists or sets, strings in Python have many powerful functions to handle text.

How to Create a String

A string in Python is a sequence of characters like "Hello" or 'Python123'

In [26…
```
name = "Python"
#name is the variable.
#"Python" is the string.
#This means you're storing the text "Python" inside the variable name.
```

Operations in Strings

Operation	Syntax / Code	Explanation
Assign a string	`name = "Python"`	Creates a string and stores it in a variable
Empty string	`empty = ""`	Creates a string with no characters
Using single quotes	`msg = 'Hello'`	You can use single quotes for strings
Using double quotes	`msg = "Hello"`	You can also use double quotes for strings
Multi-line string	`text = '''Line1\nLine2'''`	Creates a string that spans multiple lines
String from input	`name = input("Enter name: ")`	Takes string input from the user
String from number	`num_str = str(123)`	Converts a number to a string using `str()`
String concatenation	`greet = "Hello " + "World"`	Joins two strings together
String repetition	`repeat = "Hi! " * 3`	Repeats a string multiple times
String with variables	`msg = f"Hello {name}"`	Creates a string using variable inside (called f-string)
Join list of strings	`sentence = " ".join(["I", "love", "Python"])`	Joins multiple strings from a list into one string

Example Using Operations

In [28…
```
#  1. Assigning a String to a Variable

name = "Python"
```

In [29…
```
# 2. Using Single or Double Quotes
text1 = 'Hello'
text2 = "World"
```

In [31…
```
# 3. Multi-line String (Using Triple Quotes)
para = '''This is
a multi-line
string.'''
```

In [32…
```
 # 4. Empty String
empty = ""
```

In [ ]:
```
# 5. Taking String Input from the User
user_name = input("Enter your name: ")
# input() lets the user type something. It's stored as a string.
```

In [33…
```
# 6. Converting Other Types to String
num = 123
num_str = str(num)  # becomes "123"
# Use str() to convert numbers or other types into a string.
```

6. Dictionary

A dictionary stores data in **key-value pairs**. It's like a real-life dictionary, where each word (key) has a definition (value). Dictionaries are **unordered** and can be changed (mutable).

How to Create a Dictionary:

- Dictionaries are created using curly braces `{}`.
- Keys and values are separated by a colon `:`.

In [35…
```
# Creating a dictionary of student grades
student_grades = {
    "Alice": 85,
    "Bob": 92,
    "Charlie": 78
}

# Accessing values by key
print("Bob's grade:", student_grades["Bob"])
```

```
Bob's grade: 92
```

Operations in Dictionary

Operation	Code / Syntax	Explanation
Create empty dictionary	`my_dict = {}`	Creates a dictionary with no data
Create with key-value pairs	`student = {"name": "John", "age": 20}`	Creates a dictionary with 2 pairs: name & age

pairs		age
Using `dict()` function	`person = dict(name="Alice", age=25)`	Uses `dict()` constructor to create a dictionary
From list of tuples	`data = dict([("x", 1), ("y", 2)])`	Converts list of pairs into a dictionary
Add new key-value	`student["grade"] = "A"`	Adds a new item with key `grade` and value `A`
Copy dictionary	`new_dict = old_dict.copy()`	Makes a copy of an existing dictionary
Nested dictionary	`emp = {"id": 101, "info": {"name": "Raj", "age": 30}}`	A dictionary inside another dictionary
Fromkeys() method	`d = dict.fromkeys(["a", "b"], 0)`	Creates keys "a" and "b" with default value `0`
Create with mixed data types	`data = {"id": 1, "marks": [80, 90], "pass": True}`	Values can be of any data type: list, int, bool, etc.

In [ ]:
```python
# Dictionary Operations
student_grades["David"] = 88  # Adding a new student
student_grades["Alice"] = 90  # Updating a grade
del student_grades["Charlie"]  # Removing a student
print(student_grades)
```

Summary

Topics Covered:

- **What is a Data Structure?** A way to store and manage data for easy access and modification.
- **Types of Data Structures:**
  - **List** – Ordered, changeable collection.
  - **Tuple** – Ordered, unchangeable collection.
  - **Set** – Unordered, no duplicates.
  - **Dictionary** – Key-value pair storage.
- **String as a Data Structure** A sequence of characters with various operations like slicing, concatenation, etc.

Connect @
Mail (Sanjana): sanjanadevibihana@gmail.com
Contact: +91-6283762268

SANJANA DEVI

AI/ML Student | Intern @ CodroidHub Private Limited | First-Year Engineering Student.

SANJANA DEVI