ikSaan.com

Data Structures in Python

Python offers several built-in data structures that are essential for organizing, managing, and storing data in programs.

These include lists, tuples, sets, and dictionaries. Each of these structures has its unique properties and use cases.

ikSaan.com

Data Structures in Python

Lists

A list is an ordered, mutable collection of elements, which can hold items of different data types (e.g., integers, strings, objects).

Creating a List:

```python
my_list = [1, 2, 3, "hello", True]
```

ikSaan.com

Data Structures in Python

Lists

Accessing Elements: You can access list elements using indexing (starting from 0 for the first element).

Example:

```python
print(my_list[0])  # Output: 1
print(my_list[-1])  # Output: True (last element)
```

ikSaan.com

Data Structures in Python

Lists

Modifying Lists: Lists are mutable, meaning you can change, add, and remove elements.

Change an element:

Example:

```python
my_list[1] = "world"
print(my_list)  # Output: [1, "world", 3, "hello", True]
```

ikSaan.com

Data Structures in Python

Lists

Add elements:

- **append(): Adds an element to the end of the list.**
- **insert(): Adds an element at a specified index.**

```python
my_list.append("new item")
my_list.insert(2, "inserted item")
```

ikSaan.com

Data Structures in Python

Lists

Remove elements:

- **remove(): Removes the first occurrence of an item.**
- **pop(): Removes an item by index (and returns it).**
- **clear(): Removes all elements from the list.**

```python
my_list.remove(1)  # Removes the element 1
item = my_list.pop(2)  # Removes the element at index 2
```

ikSaan.com

Data Structures in Python

Lists

List Slicing:

Example:

```python
sublist = my_list[1:4]  # Gets a slice of the list from index 1 to 3
```

ikSaan.com

Data Structures in Python

Lists

List Comprehensions:

List comprehensions provide a concise way to create lists based on existing iterables.

```python
squares = [x**2 for x in range(5)]
print(squares)  # Output: [0, 1, 4, 9, 16]
```

ikSaan.com

Data Structures in Python

Tuples

A tuple is similar to a list but is immutable, meaning its elements cannot be changed after creation. They are often used for storing related but unchangeable data.

Creating a Tuple:

```python
my_tuple = (1, 2, 3, "hello", True)
```

ikSaan.com

Data Structures in Python

Tuples

Accessing Elements:

```python
print(my_tuple[0])  # Output: 1
```

Unpacking Tuples: (unpacking).

You can assign the values of a tuple directly to variables

```python
a, b, c = (1, 2, 3)
print(a, b, c)  # Output: 1 2 3
```

ikSaan.com

Data Structures in Python

Tuples

Tuple Methods:

Tuples have fewer methods compared to lists, but they do have count() (returns the number of occurrences of an element) and index() (returns the index of the first occurrence of an element).

ikSaan.com

Data Structures in Python

Sets

A set is an unordered, mutable collection of unique elements (no duplicates).

Sets are used for membership testing, removing duplicates, and performing mathematical set operations like union and intersection.

ikSaan.com

Data Structures in Python

Sets

Creating a Set:

```python
my_set = {1, 2, 3, 4}
```

ikSaan.com

Data Structures in Python

Sets

Adding and Removing Elements:

- **add(): Adds an element to the set.**
- **remove() or discard(): Removes an element from the set. discard() doesn't raise an error if the element isn't found.**

```python
my_set.add(5)
my_set.remove(2)
```

ikSaan.com

Data Structures in Python

Sets

Set Operations:

- **Union: Combines two sets (all elements).**
- **Intersection: Elements common to both sets.**
- **Difference: Elements in one set but not the other.**

```python
set1 = {1, 2, 3}
set2 = {2, 3, 4}

union_set = set1.union(set2)  # {1, 2, 3, 4}
intersection_set = set1.intersection(set2)  # {2, 3}
difference_set = set1.difference(set2)  # {1}
```

ikSaan.com

Data Structures in Python

Dictionaries

A dictionary is an unordered, mutable collection of key-value pairs. Each key must be unique, and it is used to access its corresponding value.

Creating a Dictionary:

```python
my_dict = {"name": "Alice", "age": 25, "city": "New York"}
```

ikSaan.com

Data Structures in Python

Dictionaries

Accessing and Modifying Dictionary Items:

Access a value by key:

```python
print(my_dict["name"])  # Output: Alice
```

ikSaan.com

Data Structures in Python

Dictionaries

Add or modify a key-value pair:

```python
my_dict["age"] = 26  # Modifies the value of "age"
my_dict["country"] = "USA"  # Adds a new key-value pair
```

ikSaan.com

Data Structures in Python

Dictionaries

Remove a key-value pair:

- **pop(): Removes a key-value pair and returns the value.**
- **del: Deletes a key-value pair.**
- **clear(): Removes all items from the dictionary.**

```python
age = my_dict.pop("age")  # Removes "age" and returns its value
del my_dict["city"]  # Removes "city" from the dictionary
my_dict.clear()  # Clears the entire dictionary
```

ikSaan.com

Data Structures in Python

Dictionaries

Dictionary Methods:

- keys(): Returns all keys in the dictionary.
- values(): Returns all values in the dictionary.
- items(): Returns all key-value pairs as tuples.

```python
keys = my_dict.keys()
values = my_dict.values()
pairs = my_dict.items()
```

ikSaan.com

Data Structures in Python

Dictionaries

Iterating Over Dictionaries: You can iterate over the keys, values, or both.

```python
for key in my_dict:
    print(key, my_dict[key])

for key, value in my_dict.items():
    print(f"{key}: {value}")
```

ikSaan.com

Data Structures in Python

Dictionaries

Choosing the Right Data Structure

- List: Use when you need an ordered collection that can change (e.g., an inventory list).
- Tuple: Use when you need an ordered collection that cannot change (e.g., coordinates, database rows).
- Set: Use when you need a collection of unique items (e.g., removing duplicates from a list).
- Dictionary: Use when you need to associate keys with values (e.g., a phone book or configuration settings).