Data Structures in Python

1. High-level comparison

Type	Library	Ordered	Mutable	Allows duplicates	Indexing	Typical use
list	Built-in	Yes	Yes	Yes	By integer index [i]	Generic sequence of items (mixed types)
tuple	Built-in	Yes	No	Yes	By integer index [i]	Fixed records, function returns, dictionary keys when hashable
dict (dictionary)	Built-in	Insertion-ordered (3.7+)	Yes (keys & values can be updated)	Keys: unique, Values: can repeat	By key d[key]	Key–value mappings / lookups
set	Built-in	Unordered	Yes	No (all elements unique)	No positional indexing	Membership tests, removing duplicates, set algebra
numpy.ndarray	NumPy	Yes (n-D)	Yes	Yes	By indices [i], [i, j], slicing	Fast numeric arrays, linear algebra, ML preprocessing
pandas.DataFrame	pandas	2D (rows × columns, labeled)	Yes	Column values may repeat	By labels (.loc), positions (.iloc)	Tabular data analysis, cleaning, aggregation

2. list – common methods / operations

Selected, commonly used methods (not all methods).

Method / operation	**Description**
append(x)	Add element x to the end of the list.
extend(iterable)	Append all elements from another iterable.
insert(i, x)	Insert x at position i.
remove(x)	Remove first occurrence of x (raises error if not present).
pop([i])	Remove and return element at index i (or last element if i omitted).
clear()	Remove all elements (empty list).
index(x)	Return index of first occurrence of x.
count(x)	Count occurrences of x.
sort(key=None, reverse=False)	In-place sort.
reverse()	In-place reverse of element order.
copy()	Shallow copy of the list.
Slicing lst[a:b:c]	Extract a sub-list or step through elements.
Built-ins: len, sum, min, max	Work on lists of compatible types.

3. tuple – common methods / operations

Tuples are immutable, so only a few methods.

Method / operation	Description
count(x)	Count occurrences of x.
index(x)	Index of first occurrence of x.
Indexing t[i]	Get element at position i.
Slicing t[a:b:c]	Get sub-tuple.
Concatenation t1 + t2	Returns new tuple with elements of both.
Repetition t * n	Repeat tuple n times.
Can be used as keys in dict / elements of set	If all elements are hashable.

---

4. dict (dictionary) – common methods / operations

Method / operation	Description
d[key]	Get value for key (error if missing).
get(key, default=None)	Safe get with default if key not found.
keys()	View of keys.
values()	View of values.
items()	View of (key, value) pairs.
update(other)	Merge/update with another dict or key=value pairs.
pop(key[, default])	Remove key and return value (or default).
popitem()	Remove and return last inserted (key, value).
setdefault(key, default=None)	Get value if exists, otherwise set and return default.

Method / operation	Description
clear()	Remove all items.
fromkeys(iterable, value=None)	Class method: make new dict from iterable of keys.
Membership: key in d	Test if key exists. Python documentation+1

---

5. set – common methods / operations

Method / operation	Description
add(x)	Add element x.
remove(x)	Remove x (error if not present).
discard(x)	Remove x if present (no error if missing).
pop()	Remove and return an arbitrary element.
clear()	Remove all elements.
union(other) or `	`
intersection(other) or &	Elements common to both sets.
difference(other) or -	Elements in first but not in second.
symmetric_difference(other) or ^	Elements in exactly one of the sets.
issubset(other) or <=	Test subset relation.
issuperset(other) or >=	Test superset relation.
copy()	Shallow copy.
Membership: x in s	Very fast membership test (hash-based). Python documentation+1

---

6. numpy.ndarray – common methods / operations

NumPy’s core object is ndarray (N-dimensional array). NumPy+4NumPy+4NumPy+4

Method / operation	Description
np.array(data)	Create an array from list/tuple/other sequence.
shape (attribute)	Tuple of array dimensions.
dtype (attribute)	Data type of elements.
Indexing a[i], a[i, j]	Access elements by indices.
Slicing / boolean indexing	a[1:5], a[a > 0] etc.
reshape(new_shape)	Return a reshaped view/copy.
astype(new_dtype)	Cast to another dtype.
sum(axis=None)	Sum along an axis or all elements.
mean(axis=None)	Mean along an axis.
max, min, argmax, argmin	Extremes and their indices.
transpose() or a.T	Transpose array axes.
dot(other) / @	Matrix / vector multiplication.
ravel() / flatten()	Return 1D version of array.
copy()	Explicit copy of array.

7. pandas.DataFrame – common methods / operations

A DataFrame is a 2D, labeled, tabular data structure.

Method / operation	**Description**
Construction: pd.DataFrame(data, ...)	Create from dicts, arrays, other DataFrames, etc.
head(n) / tail(n)	First / last n rows.
info()	Summary of columns, dtypes, non-null counts.
describe()	Descriptive statistics for numeric columns.
Column access: df["col"], df[["c1","c2"]]	Select one or more columns.
Row/label access: df.loc[row_sel, col_sel]	Select by labels / boolean masks.
Positional access: df.iloc[row_idx, col_idx]	Select by integer positions.
assign(...)	Add new columns (returns new DataFrame).
drop(labels, axis=0/1)	Drop rows or columns.
sort_values(by=...)	Sort by one or more columns.
groupby(cols)	Group rows, then aggregate (.sum(), .mean(), ...).
merge, join, concat	Combine DataFrames (relational joins / concatenations).
fillna(value) / dropna()	Handle missing values.
to_csv, read_csv	Save/load from CSV; similar methods for Excel, SQL, etc.

8. Conceptual differences (summary)

- **Mutability and safety**
  - Use **list** when you plan to change (append, remove) elements frequently.
  - Use **tuple** when data is fixed (e.g., coordinates, constant configuration), or when you need an immutable, hashable object (for dict keys or set elements).
- **Structure of data**
  - Use **dict** for lookups by name/key (e.g., student_id → record).
  - Use **set** when you only care about uniqueness and membership (e.g., distinct labels, removing duplicates).
- **Numerical computing vs general Python**
  - Use **NumPy arrays** (ndarray) for numeric, homogeneous data where you need vectorized operations, linear algebra, or speed (e.g., ML preprocessing, image data).
  - Use **pandas DataFrame** for labeled, heterogeneous, tabular data (data analysis, time series, CSV files).

Coding Examples

1. list

Common list methods and examples

Method / operation	Description	Example

append(x)	Add x to end of list	nums = [1, 2, 3]
		nums.append(4)
		print(nums)

extend(iterable)	Add all elements from iterable	nums = [1, 2]
		nums.extend([3, 4])
		print(nums)

insert(i, x)	Insert x at position i	nums = [1, 3, 4]
		nums.insert(1, 2)
		print(nums) # [1, 2, 3, 4]

remove(x)	Remove first occurrence of x	nums = [1, 2, 2,3]
		nums.remove(2)
		print(nums) # [1, 2, 3]

pop([i])	Remove and return item at i (last if omitted)	nums = [10, 20, 30]
		last = nums.pop()
		print(last, nums) # 30 [10, 20]

clear()	Remove all elements	nums = [1, 2, 3]
		nums.clear()
		print(nums)

index(x)	Return index of first x	nums = [5, 10, 15]

Method / operation	Description	Example
		print(nums.index(10)) #
count(x)	Count occurrences of x	nums = [1, 1, 2] print(nums.count(1)) # 2

sort(key=None, reverse=False)	Sort list in place	nums = [3, 1, 2]
		nums.sort()
		print(nums) # [1, 2, 3]\n

reverse()	Reverse in place	nums = [1, 2, 3]
		nums.reverse()
		print(nums) # [3, 2, 1]\n

copy()	Shallow copy	nums = [1, 2]
		nums2 = nums.copy()
		print(nums2) # [1, 2]\n

Slicing lst[a:b:c]	Get sub-list	nums = [0, 1, 2, 3, 4] print(nums[1:4]) # [1, 2, 3] print(nums[::2]) # [0, 2, 4]
Built-ins: len(lst)	Number of elements	nums = [1, 2, 3] print(len(nums)) # 3\n

2. tuple

Common tuple operations and examples

Method / operation	Description	Example
count(x)	Count occurrences of x	t = (1, 2, 2, 3) print(t.count(2)) # 2\n
index(x)	Index of first occurrence of x	t = (10, 20, 30) print(t.index(20)) # 1\n
Indexing t[i]	Access element by index	t = ('a', 'b', 'c') print(t[1]) # 'b'\n

Slicing t[a:b:c]	Sub-tuple	t = (0, 1, 2, 3)
		print(t[1:3]) # (1, 2)

Concatenation t1 + t2	Join tuples	a = (1, 2)
		b = (3, 4)
		print(a + b) # (1, 2, 3, 4)

Repetition t * n	Repeat tuple n times	t = (1, 2)
		print(t * 3) # (1, 2, 1, 2, 1, 2)

Note: tuples are **immutable**, so there is no append, remove, etc.

3. dict (dictionary)

Common dictionary methods and examples

Method / operation	Description	Example
d[key]	Get value for key (error if not found)	student = {'name': 'Ali', 'age': 20} print(student['name']) # 'Ali'
get(key, default=None)	Safe get with default	print(student.get('grade', 'N/A')) # 'N/A'
keys()	View of keys	print(list(student.keys())) # ['name', 'age']
values()	View of values	print(list(student.values())) # ['Ali', 20]
items()	View of (key, value) pairs	for k, v in student.items(): print(k, v)

update(other)	Update with another dict or key=value	student.update({'age': 21, 'grade': 'A'})
		print(student)

pop(key[, default])	Remove and return value for key	age = student.pop('age') print(age) # 21 print(student) # no 'age'\n

popitem()	Remove and return last inserted item	item = student.popitem()
		print(item) # e.g. ('grade', 'A')

setdefault(key, default)	Get value or set default	student.setdefault('city', 'Tanta') print(student['city']) # 'Tanta'
clear()	Remove all items	student.clear()\nprint(student) # {}\n

fromkeys(iterable, value)	Create dict from keys	keys = ['a', 'b', 'c']
		d = dict.fromkeys(keys, 0)
		print(d) # {'a': 0, 'b': 0, 'c': 0}

Membership key in d	Test existence of key	print('name' in student) # False (after clear)

4. set

Common set methods and examples

Method / operation	Description	Example

add(x)	Add element x	s = {1, 2}
		s.add(3)
		print(s) # {1, 2, 3}\n

remove(x)	Remove x (error if missing)	s.remove(2) print(s) # {1, 3}\n
discard(x)	Remove x if present, no error	s.discard(5) # no error\n

pop()	Remove and return arbitrary element	val = s.pop()
		print(val, s)

clear()	Remove all elements	s.clear()
		print(s) # set()

union(other) / `	`	All elements from both sets
intersection(other) / &	Common elements	print(a & b) # {2} print(a.intersection(b)) # {2}\n

difference(other) / -	Elements in a not in b	print(a - b) # {1}
		print(a.difference(b)) # {1}

symmetric_difference(other) / ^	Elements in exactly one set	print(a ^ b) # {1, 3}
		print(a.symmetric_difference(b)) # {1, 3}

issubset(other) / <=	Test subset	small = {1, 2}
		large = {1, 2, 3}

Method / operation	Description	Example
		print(small.issubset(large)) # True
issuperset(other) / >=	Test superset	print(large.issuperset(small)) # True

copy()	Shallow copy	c = large.copy()
		print(c) # {1, 2, 3}

Membership x in s	Test existence	print(2 in large) # True

5. numpy.ndarray

import numpy as np

Important ndarray attributes / methods / operations

Attribute / method	Description	Example
np.array(data)	Create array	a = np.array([1, 2, 3]) print(a) # [1 2 3]
shape	Dimensions of array	print(a.shape) # (3,)
dtype	Data type of elements	print(a.dtype) # e.g. int64
Indexing a[i], a[i,j]	Element access	b = np.array([[1, 2], [3, 4]]) print(b[0, 1]) # 2
Slicing / boolean indexing	Sub-arrays / conditions	print(b[:, 0]) # first column -> [1 3] print(b[b > 2]) # elements > 2 -> [3 4]
reshape(new_shape)	Change shape	c = np.arange(6) print(c.reshape(2, 3))# [[0 1 2]\n# [3 4 5]]
astype(dtype)	Convert data type	f = a.astype(float)

Attribute / method	Description	Example
		print(f, f.dtype)
sum(axis=None)	Sum elements	print(b.sum()) # 10 print(b.sum(axis=0)) # column-wise [4 6]
mean(axis=None)	Mean value	print(b.mean()) # 2.5
max, min	Extremes	print(b.max(), b.min()) # 4 1
argmax, argmin	Indices of extremes	print(b.argmax()) # 3 (flat index)
transpose() or .T	Transpose	print(b.T)\n# [[1 3]\n# [2 4]]
dot(other) or @	Matrix / vector product	x = np.array([1, 2])\ny = np.array([3, 4])\nprint(x @ y) # 11
ravel() / flatten()	1D view / copy	print(b.ravel()) # [1 2 3 4]
copy()	Explicit copy	b2 = b.copy()\n

---

6. pandas.DataFrame

Assume:

import pandas as pd

Important DataFrame methods / operations

Method / operation	**Description**	**Example**

pd.DataFrame(data)	Construct DataFrame	data = {'name': ['Ali', 'Sara'], 'age': [20, 22]}
		df = pd.DataFrame(data)
		print(df)

head(n) / tail(n)	First / last n rows	print(df.head(1))
info()	Summary of dtypes and non-null counts	df.info()
describe()	Descriptive stats	print(df.describe())
Column selection df['col']	Select single column (Series)	print(df['age'])
Multiple columns df[['c1','c2']]	Select subset of columns	print(df[['name', 'age']])
loc[row_sel, col_sel]	Label-based selection	print(df.loc[0, 'name']) # 'Ali' print(df.loc[:, 'age']) # all ages
iloc[row_idx, col_idx]	Position-based selection	print(df.iloc[0, 1]) # 20

assign(new_col=...)	Add new column (returns new df)	df2 = df.assign(age_plus_1=df['age'] + 1)
		print(df2)

drop(labels, axis=0/1)	Drop rows / columns	# drop column 'age'
		df_no_age = df.drop('age', axis=1)

Method / operation	**Description**	**Example**
sort_values(by=...)	Sort by column(s)	sorted_df = df.sort_values(by='age', ascending=False)\nprint(sorted_df)

groupby(cols) + aggregation	Group rows and aggregate	df = pd.DataFrame({'city': ['A', 'A', 'B'], 'score': [10, 20, 30]})
		print(df.groupby('city')['score'].mean())

merge, join, concat	Combine DataFrames	df1 = pd.DataFrame({'id': [1, 2], 'val': [10, 20]})\ndf2 = pd.DataFrame({'id': [1, 2], 'name': ['Ali', 'Sara']})
		merged = pd.merge(df1, df2, on='id')
		print(merged)

fillna(value)	Replace missing values	df3 = pd.DataFrame({'x': [1, None, 3]})
		print(df3.fillna(0))

dropna()	Drop rows with missing values	print(df3.dropna())

to_csv(path) / read_csv(path)	Save/load CSV	df.to_csv('students.csv', index=False)
		loaded = pd.read_csv('students.csv')

Table: When to Use Each Data Structure in Python

Data Structure	When to Use It	Best For	Avoid When
List	You need an **ordered**, **changeable**, and **indexed** collection of items.	General-purpose storage, dynamic arrays, stacks, queues. Mixed-type elements.	When you need fast numeric computation, or when structure must be immutable.
Tuple	You want **fixed-size**, **immutable** ordered data. Useful for ensuring data cannot be modified.	Function return values (multiple outputs), coordinates, database records, using as dictionary keys.	When you need to frequently add/remove elements.
Dictionary	You need **key–value mapping** with fast lookup, insertion, and deletion. Keys must be unique.	Storing data by name (e.g., student["age"]), configuration settings, JSON-like structures.	When order is important for numeric operations, or when duplicates are needed.
Set	You need a collection of **unique**, **unordered**, and **non-duplicate** items.	Removing duplicates, membership tests, set algebra (union, intersection).	When element order matters, or when storing mutable/unhashable objects (lists, dicts).
NumPy Array (ndarray)	You work with **numerical data** and need **high performance**, **vectorized operations**, **matrices**, or **linear algebra**.	Machine learning, scientific computing, image data, matrix calculations.	Storing mixed data types, labels, or non-numeric data.
Pandas DataFrame	You need to work with **tabular data** (rows × columns) with **labels**,	Data analysis, CSV/Excel datasets,	Heavy numeric operations (use NumPy) or simple

Data Structure	When to Use It	Best For	Avoid When
	missing values, **statistics**, and **data cleaning**.	grouping, merging, time series.	lists/dicts where DataFrame is unnecessary.

Short Decision Guide (Fast Rules)

Situation	Recommended Structure
Need ordered and mutable collection	**List**
Need fixed structure that must not change	**Tuple**
Need fast lookups by key/name	**Dictionary**
Need unique items, no duplicates	**Set**
Need fast numeric computations	**NumPy Array**
Need data analysis on tables	**Pandas DataFrame**
Need to remove duplicates from a list	**Set** (then convert back to list if needed)
Need to return multiple values from a function	**Tuple**
Need to store student data (name, age, grade)	**Dictionary**
Need to store multiple rows of student records	**Pandas DataFrame**

Expanded Explanation Table (More Detailed)

Data Structure	Ordered	Mutable	Duplicate Allowed	Fast Lookup	Typical Data	Memory Usage	Use Case Examples
List	Yes	Yes	Yes	Slow for search	Mixed types	Medium	Tasks, items, sequences, dynamic data

Data Structure	Ordered	Mutable	Duplicate Allowed	Fast Lookup	Typical Data	Memory Usage	Use Case Examples
Tuple	Yes	No	Yes	Fast (immutable)	Fixed records	Low	Coordinates, settings, function return values
Dictionary	Yes (insertion order)	Yes	Keys: No, Values: Yes	Fast (O(1) avg)	Key → value pairs	Medium	JSON objects, configs, mapping ids to data
Set	No	Yes	No	Very fast membership	Unique items	Low	Remove duplicates, mathematical sets
NumPy Array	Yes	Yes	Yes	Very fast vectorized operations	Numeric data	Very Low	Machine learning, matrices, images
DataFrame	Yes	Yes	Yes	Fast column operations	Tabular data	High	CSVs, analytics, cleaning, statistics

---

If you want, I can also prepare: