**Python Data Structures Cheat Sheet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Ordered | Mutable | Allows Duplicates | Syntax |
| List | ✅ | ✅ | ✅ | [1,2,3] |
| Tuple | ✅ | ❌ | ✅ | (1,2,3) |
| Set | ❌ | ✅ | ❌ | {1,2,3} |
| Dict | ❌ | ✅ | Keys ❌ | {'a':1} |
| String | ✅ | ❌ | ✅ | "abc" |
| frozenset | ❌ | ❌ | ❌ | frozenset([...]) |

1. **List**

Ordered, mutable, allows duplicates

Create

fruits = ["apple", "banana", "cherry"]

Access

fruits[0] # 'apple'

fruits[-1] # 'cherry'

Modify

fruits[1] = "orange" # ['apple', 'orange', 'cherry']

Add / Remove

fruits.append("mango") # Add at end

fruits.insert(1, "grape") # Insert at index

fruits.extend(["kiwi", "melon"]) # Merge lists

fruits.remove("apple") # Remove by value

fruits.pop() # Remove last element

fruits.pop(1) # Remove by index

del fruits[0] # Delete index 0

fruits.clear() # Empty list

Search / Count

fruits = ["apple", "banana", "apple"]

fruits.index("banana") # 1

fruits.count("apple") # 2

Sort / Reverse

nums = [5, 2, 9, 1]

nums.sort() # [1, 2, 5, 9]

nums.sort(reverse=True) # [9, 5, 2, 1]

nums.reverse() # Reverse order

Copy

a = [1, 2, 3]

b = a.copy() # Shallow copy

List Comprehension

squares = [x\*\*2 for x in range(5)] # [0, 1, 4, 9, 16

1. **Tuple**

Ordered, immutable, allows duplicates

Create

t = (1, 2, 3)

single = (5,) # Single element tuple

Access

t[0] # 1

Count / Index

t.count(2) # 1

t.index(3) #

Unpack

a, b, c = t # a=1, b=2, c=3

1. **Set**

Unordered, mutable, unique values only

Create  
 s = {1, 2, 3, 4}

Add / Remove

s.add(5)

s.update([6, 7]) # Add multiple

s.remove(3) # Remove; error if not exist

s.discard(10) # Remove; no error if not exist

s.pop() # Remove random element

s.clear() # Empty set

Set Operations

a = {1, 2, 3}

b = {3, 4, 5}

a | b # Union -> {1,2,3,4,5}

a & b # Intersection -> {3}

a - b # Difference -> {1,2}

a ^ b # Symmetric diff -> {1,2,4,5}

Relations

a.issubset(b)

a.issuperset(b)

a.isdisjoint(b)

**4. Dictionary**

Key–Value pairs, unordered, mutable

Create

person = {"name": "Alice", "age": 25, "city": "Paris"}

Access

person["name"] # 'Alice'

person.get("age") # 25

Add / Update

person["email"] = "alice@mail.com"

person.update({"age": 26})

Remove

person.pop("city") # Remove by key

person.popitem() # Remove last added pair

del person["name"]

person.clear()

Keys / Values / Items

person.keys() # dict\_keys(['age', 'email'])

person.values() # dict\_values([26, 'alice@mail.com'])

person.items() # dict\_items([('age', 26), ('email', 'alice@mail.com')])

Dictionary Comprehension

squares = {x: x\*x for x in range(5)} # {0:0,1:1,2:4,3:9,4:16}

1. **String**

Immutable, sequence of characters

Create

s = "Hello World"

Access

s[0] # 'H'

s[-1] # 'd'

Common Methods

s.lower() # 'hello world'

s.upper() # 'HELLO WORLD'

s.title() # 'Hello World'

s.strip() # Removes whitespace

s.replace("World","Python")

s.split() # ['Hello', 'World']

'-'.join(['a','b','c']) # 'a-b-c'

s.startswith("He") # True

s.endswith("ld") # True

Find / Count

s.find("lo") # 3

s.count("l") # 3

6. Array (from array module)

More memory-efficient for large numeric lists

import array

arr = array.array('i', [1, 2, 3, 4])

arr.append(5)

arr.remove(2)

arr.reverse()

**7. Stack (LIFO)**

Using list or collections.deque

stack = []

stack.append(1)

stack.append(2)

stack.pop() # 2

Or:

from collections import deque

stack = deque()

stack.append('a')

stack.append('b')

stack.pop() # 'b'

**8. Queue (FIFO)**

Using collections.deque

from collections import deque

queue = deque(["a", "b", "c"])

queue.append("d") # Enqueue

queue.popleft() # Dequeue -> 'a

**9. Priority Queue / Heap**

import heapq

nums = [5, 1, 9, 3]

heapq.heapify(nums)

heapq.heappush(nums, 0)

heapq.heappop(nums)

**10. Collections Module Extras**

Counter

from collections import Counter

c = Counter("banana")

# Counter({'a': 3, 'n': 2, 'b': 1})

c.most\_common(1) # [('a', 3)]

OrderedDict

from collections import OrderedDict

od = OrderedDict()

od["a"] = 1

od["b"] = 2

defaultdict

from collections import defaultdict

d = defaultdict(int)

d["x"] += 1

namedtuple

from collections import namedtuple

Point = namedtuple('Point', 'x y')

p = Point(10, 20)

p.x, p.y # (10, 20)

deque

from collections import deque

dq = deque([1, 2, 3])

dq.appendleft(0)

dq.extend([4,5])

dq.rotate(1) # Rotate right by 1

**11. frozenset**  
 Immutable set

fs = frozenset([1, 2, 3])

**12. Useful Built-in Functions (All Structures)**

|  |  |  |
| --- | --- | --- |
| Function | Description | Example |
| len() | Count items | len([1,2,3]) → 3 |
| max() / min() | Find largest/smallest | max({2,5,1}) → 5 |
| sum() | Sum all | sum([1,2,3]) → 6 |
| sorted() | Returns sorted list | sorted({3,1,2}) → [1,2,3] |
| any() | True if any True | any([0, False, 5]) → True |
| all() | True if all True | all([1, True, 5]) → True |
| enumerate() | Index + Value | for i,v in enumerate(['a','b']) |
| zip() | Combine iterables | list(zip([1,2],[3,4])) → [(1,3),(2,4)] |
| map() | Apply function | map(str.upper, ['a','b']) → ['A','B'] |
| filter() | Filter items | filter(lambda x:x>2,[1,2,3]) → [3] |
| reduce() | Cumulative operation | *(from functools)* |

**14. Itertools Module (Advanced Tools)**

import itertools as it

list(it.permutations([1,2,3])) # All orderings

list(it.combinations([1,2,3],2)) # Pairs

list(it.product([1,2], repeat=2)) # Cartesian product

list(it.chain([1,2],[3,4])) # Combine iterables