Python Syntax Quick Reference - Data Structures & Built-ins

ARRAYS (Lists)

Creation & Basic Operations

```
# Initialize
arr = []
arr = [0] * n  # n zeros
arr = [i for i in range(n)]  # 0 to n-1
arr = [[0] * cols for _ in range(rows)]  # 2D array

# Add elements
arr.append(x)  # Add to end - O(1)
arr.insert(i, x)  # Insert at index - O(n)
arr.extend([1, 2, 3])  # Add multiple - O(k)

# Remove elements
arr.pop()  # Remove last - O(1)
arr.pop(i)  # Remove at index - O(n)
arr.remove(x)  # Remove first occurrence of x - O(n)
del arr[i]  # Delete at index - O(n)

# Access
arr[i]  # Get element
arr[-1]  # Last element
arr[-2]  # Second to last

# Slicing
arr[start:end]  # From start to end-1
arr[start:]  # From start to end
arr[:end]  # From beginning to end-1
arr[:]  # Copy entire list
arr[::2]  # Every other element
arr[::-1]  # Reverse

# Common methods
len(arr)  # Length
max(arr), min(arr)  # Max/min
```

```
sum(arr)  # Sum
arr.index(x)  # First index of x
arr.count(x)  # Count occurrences
arr.sort()  # Sort in-place
sorted(arr)  # Return sorted copy
arr.reverse()  # Reverse in-place
```

List Comprehensions

```
# Basic
squares = [x**2 for x in range(10)]

# With condition
evens = [x for x in range(10) if x % 2 == 0]

# Nested
matrix = [[i*j for j in range(5)] for i in range(5)]

# Flatten
flattened = [item for sublist in matrix for item in sublist]
```

STRINGS

Creation & Operations

```
# Create
s = "hello"
s = str(123)  # "123"
s = ''.join(['a', 'b', 'c'])  # "abc"

# Common operations
s.lower(), s.upper()
s.strip()  # Remove whitespace
s.split()  # Split by whitespace
s.split(',')  # Split by delimiter
s.replace('old', 'new')
s.startswith('h'), s.endswith('o')
s.isalpha(), s.isdigit(), s.isalnum()

# Substring
s[start:end]
s.find('ll')  # Returns index or -1
'll' in s  # Returns boolean
```

```
# Format
f"Value: {x}"
"Value: {}".format(x)
```

String Manipulation

```
# Reverse
reversed_s = s[::-1]

# Sort characters
sorted_s = ''.join(sorted(s))

# Count characters
from collections import Counter
char_count = Counter(s)

# Check palindrome
is_palindrome = s == s[::-1]
```

HASH TABLES (Dictionaries)

Dictionary Operations

```
# Create
d = {}
d = dict()
d = {'a': 1, 'b': 2}
d = dict(zip(keys, values))

# Access
d['key']  # Raises KeyError if not exists
d.get('key')  # Returns None if not exists
d.get('key', default_value)  # Returns default if not exists

# Add/Update
d['key'] = value
d.setdefault('key', default)  # Set if not exists

# Remove
del d['key']
d.pop('key')  # Returns value
d.pop('key', default)  # Returns default if not exists
```

```
# Check existence
'key' in d
'key' not in d

# Iterate
for key in d:
    print(key, d[key])

for key, value in d.items():
    print(key, value)

for value in d.values():
    print(value)

# Get all keys/values
d.keys()
d.values()
d.items()
```

DefaultDict

```
from collections import defaultdict

# Auto-initialize missing keys
d = defaultdict(int)  # Default to 0
d = defaultdict(list)  # Default to []
d = defaultdict(set)  # Default to set()

# Usage
d['new_key'] += 1  # Works even if key doesn't exist
d['new_key'].append(value)  # Works with list default
```

Counter

```
from collections import Counter

# Count occurrences
count = Counter([1, 2, 2, 3, 3, 3])
# Counter({3: 3, 2: 2, 1: 1})

count = Counter("hello")
# Counter({'l': 2, 'h': 1, 'e': 1, 'o': 1})
# Common operations
count.most_common(2)  # Top 2 most common
count['x']  # Get count (0 if not exists)
count.update([1, 2, 3])  # Add more elements
count1 + count2  # Combine counters
count1 - count2  # Subtract counts
```

SETS

Set Operations

```
# Create
s = set()
s = {1, 2, 3}
s = set([1, 2, 3])

# Add/Remove
s.add(x)
s.remove(x)  # Raises KeyError if not exists
s.discard(x)  # No error if not exists
s.pop()  # Remove arbitrary element

# Set operations
s1 | s2  # Union
s1 & s2  # Intersection
s1 - s2  # Difference
s1 ^ s2  # Symmetric difference

# Check
x in s
s1.issubset(s2)
s1.issuperset(s2)

# Size
len(s)

# Convert
list(s)  # To list
```

STACKS

Using List as Stack

```
stack = []

# Push
stack.append(x)

# Pop
if stack:
    x = stack.pop()

# Peek
if stack:
    top = stack[-1]

# Check empty
is_empty = len(stack) == 0
is_empty = not stack  # Pythonic way

# Size
size = len(stack)
```

QUEUES

Using deque (Efficient)

```
from collections import deque

# Create
queue = deque()
queue = deque([1, 2, 3])

# Enqueue
queue.append(x)  # Add to right

# Dequeue
if queue:
    x = queue.popleft()  # Remove from left

# Peek
if queue:
    front = queue[0]

# Size
size = len(queue)
```

Priority Queue (Heap)

```
import heapq

# Min heap (default)
heap = []
heapq.heappush(heap, x)
x = heapq.heappop(heap)
smallest = heap[0]  # Peek

# Initialize from list
heap = [3, 1, 4, 1, 5]
heapq.heapify(heap)

# Max heap (negate values)
heap = []
heapq.heappush(heap, -x)
x = -heapq.heappop(heap)

# K largest/smallest
k_largest = heapq.nlargest(k, arr)
k_smallest = heapq.nsmallest(k, arr)

# Heap with tuples (priority, value)
heapq.heappush(heap, (priority, value))
priority, value = heapq.heappop(heap)
```

USEFUL BUILT-IN FUNCTIONS

Math

```
abs(x)  # Absolute value
max(a, b, c), min(a, b, c)
pow(x, y)  # x^y
divmod(a, b)  # Returns (quotient, remainder)

import math
math.ceil(x)  # Round up
math.floor(x)  # Round down
math.sqrt(x)
math.gcd(a, b)  # Greatest common divisor
math.inf  # Infinity
```

Iteration

```
# Enumerate (index + value)
for i, val in enumerate(arr):
    print(i, val)

for i, val in enumerate(arr, start=1):  # Start from 1
    print(i, val)

# Zip (combine iterables)
for a, b in zip(list1, list2):
    print(a, b)

# Range
range(n)  # 0 to n-1
range(start, end)  # start to end-1
range(start, end, step)

# Reversed
for item in reversed(arr):
    print(item)

# Sorted with custom key
sorted(arr, key=lambda x: x[1])  # Sort by second element
sorted(arr, reverse=True)  # Descending
```

Any/All

```
any([True, False, False])  # True if any is True
all([True, True, True])  # True if all are True

# With generator
any(x > 5 for x in arr)
all(x > 0 for x in arr)
```

Map/Filter

```
# Map (apply function to all)
squared = list(map(lambda x: x**2, arr))

# Filter (keep only if condition)
evens = list(filter(lambda x: x % 2 == 0, arr))
```

COMMON TRICKS

Swap Variables

```
a, b = b, a
```

Multiple Assignment

```
a, b, c = 1, 2, 3
a = b = c = 0
```

Unpacking

```
first, *middle, last = [1, 2, 3, 4, 5]
# first=1, middle=[2,3,4], last=5
```

Ternary Operator

```
value = x if condition else y
```

Short-circuit Evaluation

```
result = x or default_value
result = x and y
```

In-place Operations

```
x += 1  # Faster than x = x + 1
x //= 2  # Integer division
```

Integer Division & Modulo

```
quotient = a // b
remainder = a % b
quotient, remainder = divmod(a, b)  # Both at once
```

Bit Operations

```
x & y   # AND
x | y   # OR
x ^ y   # XOR
~x      # NOT
x << n  # Left shift
x >> n  # Right shift
```

String Multiplication

```
s = 'a' * 5  # 'aaaaa'
arr = [0] * n  # [0, 0, 0, ..., 0]
```

Check Multiple Conditions

```
if x in [1, 2, 3, 4, 5]:
    pass

if 1 <= x <= 10:
    pass
```

Float to Int

```
int(3.7)  # 3 (truncate)
round(3.7)  # 4 (round)
```

ASCII / CHARACTER OPERATIONS

```
# Character to ASCII
ord('a')  # 97
ord('A')  # 65
ord('0')  # 48

# ASCII to character
chr(97)  # 'a'
chr(65)  # 'A'

# Check if character
c.isalpha()  # Letter
c.isdigit()  # Digit
c.isalnum()  # Letter or digit
c.islower()  # Lowercase
```

```
c.isupper()  # Uppercase
```

INPUT/OUTPUT FOR HACKERRANK

Fast Input

```
import sys
input = sys.stdin.readline

# Read single line
line = input().strip()

# Read integers
n = int(input())
arr = list(map(int, input().split()))

# Read multiple lines
lines = sys.stdin.readlines()
```

Output

```
print(value)
print(value, end='')  # No newline
print(value, end=' ')  # Space instead
print(*arr)  # Print list elements separated by space
print('\n'.join(map(str, arr)))  # Each on new line
```

GOTCHAS TO REMEMBER

```
# Shallow vs Deep copy
arr2 = arr  # Reference (modifying arr2 modifies arr)
arr2 = arr[:]  # Shallow copy
arr2 = arr.copy()  # Shallow copy
from copy import deepcopy
arr2 = deepcopy(arr)  # Deep copy (for nested structures)

# 2D array initialization (WRONG)
arr = [[0] * cols] * rows  # DON'T DO THIS - all rows are same reference

# 2D array initialization (CORRECT)
arr = [[0] * cols for _ in range(rows)]
```

```
# String immutability
s[0] = 'a'  # ERROR - strings are immutable
s = 'a' + s[1:]  # OK - create new string

# Dictionary iteration order
# In Python 3.7+, dictionaries maintain insertion order
```

COMMON EDGE CASES TO CHECK

```
# Empty input
if not arr:
    return result

# Single element
if len(arr) == 1:
    return arr[0]

# Negative numbers
if x < 0:
    handle_negative()

# Duplicates
if len(arr) != len(set(arr)):
    has_duplicates()

# Out of bounds
if 0 <= i < len(arr):
    safe_access()

# Division by zero
if denominator != 0:
    result = numerator / denominator
```