🐍 Python Notes: Strings, Lists, and Tuples (Complete Reference)

---

1. STRINGS IN PYTHON

1.1 Definition

A **string** in Python is a **sequence of characters** enclosed within single (`' '`) or double quotes (`" "`).
They are **immutable**, meaning once created, they cannot be modified.

```
s1 = 'Hello'
s2 = "Python"
```

1.2 String Creation & Access

```
s = "Python Programming"
print(s[0])       # Access character at index 0 → P
print(s[-1])      # Last character → g
print(s[0:6])     # Slicing → Python
```

1.3 Immutability

Strings cannot be changed in place:

```
s = "Hello"
# s[0] = 'Y' → ✗ Error: 'str' object does not support item assignment
s = 'Y' + s[1:]   # ✓ Correct way → 'Yello'
```

1.4 String Operators

Operator	Description	Example	Output
`+`	Concatenation	`'Py' + 'thon'`	`'Python'`
`*`	Repetition	`'Hi ' * 3`	`'Hi Hi Hi '`
`in`	Membership test	`'a' in 'cat'`	`True`
`not in`	Non-membership	`'z' not in 'cat'`	`True`
`[]`	Indexing	`'Hello'[1]`	`'e'`
`[start:end:step]`	Slicing	`'Python'[::2]`	`'Pto'`

1.5 String Functions (Complete Reference)

Case Conversion

Function	Description	Example	Output
`upper()`	Converts to uppercase	`'python'.upper()`	`'PYTHON'`
`lower()`	Converts to lowercase	`'PyThOn'.lower()`	`'python'`
`title()`	Capitalizes each word	`'python programming'.title()`	`'Python Programming'`
`capitalize()`	Capitalizes first letter	`'python'.capitalize()`	`'Python'`
`swapcase()`	Swaps case	`'PyThOn'.swapcase()`	`'pYtHoN'`
`casefold()`	Aggressive lowercase (for comparison)	`'ß'.casefold()`	`'ss'`

Alignment and Formatting

Function	Description	Example	Output
`center(width, char)`	Centers string	`'Hi'.center(10, '*')`	`'****Hi****'`
`ljust(width, char)`	Left aligns	`'Hi'.ljust(10, '-')`	`'Hi--------'`
`rjust(width, char)`	Right aligns	`'Hi'.rjust(10, '-')`	`'--------Hi'`
`zfill(width)`	Pads with zeros	`'45'.zfill(5)`	`'00045'`
`format()`	String formatting	`"Name: {}".format('John')`	`'Name: John'`
`f-string`	Modern formatting	`f"Hello {name}"`	`'Hello John'`

Search and Replace

Function	Description	Example	Output
`find(sub)`	Index of first occurrence	`'banana'.find('na')`	`2`
`rfind(sub)`	Last occurrence index	`'banana'.rfind('na')`	`4`
`index(sub)`	Like find(), raises error if not found	`'banana'.index('na')`	`2`
`rindex(sub)`	Reverse index	`'banana'.rindex('na')`	`4`
`count(sub)`	Count occurrences	`'banana'.count('a')`	`3`
`replace(old, new, count)`	Replace substring	`'banana'.replace('na', 'xy')`	`'baxyxy'`

Testing and Checking

Function	Description	Example	Output
`startswith(prefix)`	True if string starts with prefix	`'hello'.startswith('he')`	`True`
`endswith(suffix)`	True if ends with suffix	`'hello'.endswith('lo')`	`True`
`isalpha()`	Only letters	`'abc'.isalpha()`	`True`
`isdigit()`	Only digits	`'123'.isdigit()`	`True`
`isalnum()`	Letters or digits	`'abc123'.isalnum()`	`True`
`isspace()`	Only spaces	`' '.isspace()`	`True`
`islower()`	All lowercase	`'hello'.islower()`	`True`
`isupper()`	All uppercase	`'HELLO'.isupper()`	`True`
`istitle()`	Title case	`'Hello World'.istitle()`	`True`

Splitting and Joining

Function	Description	Example	Output
`split(sep)`	Split into list	`'a,b,c'.split(',')`	`['a', 'b', 'c']`
`rsplit(sep)`	Split from right	`'a,b,c'.rsplit(',', 1)`	`['a,b', 'c']`
`splitlines()`	Split at newlines	`'A\nB'.splitlines()`	`['A', 'B']`
`join(iterable)`	Join iterable into string	`'-'.join(['a','b','c'])`	`'a-b-c'`
`partition(sep)`	Split into 3 parts	`'abc'.partition('b')`	`('a', 'b', 'c')`
`rpartition(sep)`	Split from right	`'abcabc'.rpartition('b')`	`('abca', 'b', 'c')`

Whitespace Handling

Function	Description	Example	Output
`strip()`	Removes both sides whitespace	`' hi '.strip()`	`'hi'`
`lstrip()`	Removes left spaces	`' hi'.lstrip()`	`'hi'`
`rstrip()`	Removes right spaces	`'hi '.rstrip()`	`'hi'`

1.6 Advanced Examples

```
# Palindrome Check
s = "madam"
print(s == s[::-1])  # True

# Count vowels
vowels = "aeiou"
s = "Python Programming"
print(sum(1 for c in s.lower() if c in vowels))  # 4

# Word frequency
sentence = "this is a test this is"
print({word: sentence.split().count(word) for word in set(sentence.split())})
```

---

2. LISTS IN PYTHON

2.1 Definition

A **list** is an **ordered, mutable** collection of elements enclosed in `[]`.
Lists can hold **heterogeneous data** (mixed types).

```
nums = [10, 20, 30, 40]
mixed = [1, "hello", 3.14, True]
```

2.2 Accessing Elements

```
nums[0]     # 10
nums[-1]    # 40
nums[1:3]   # [20, 30]
```

2.3 List Functions (Complete)

Adding / Removing Elements

Function	Description	Example	Output
`append(x)`	Add to end	`[1,2].append(3)`	`[1,2,3]`
`extend(iterable)`	Add multiple	`[1,2].extend([3,4])`	`[1,2,3,4]`
`insert(i, x)`	Insert at index	`a.insert(1, 'x')`	`['a','x','b']`
`remove(x)`	Remove first match	`[1,2,3,2].remove(2)`	`[1,3,2]`
`pop(i)`	Remove & return item	`a.pop(1)`	Returns removed item
`clear()`	Remove all elements	`a.clear()`	`[]`

Sorting / Reversing

Function	Description	Example	Output
`sort()`	Sort ascending	`[3,1,2].sort()`	`[1,2,3]`
`sort(reverse=True)`	Descending	`[1,3,2].sort(reverse=True)`	`[3,2,1]`
`sorted(list)`	Returns sorted copy	`sorted([3,1,2])`	`[1,2,3]`
`reverse()`	Reverse list	`[1,2,3].reverse()`	`[3,2,1]`

Searching / Counting

Function	Description	Example	Output
`index(x)`	First index of x	`[1,2,3].index(2)`	`1`
`count(x)`	Count occurrences	`[1,2,2,3].count(2)`	`2`

Copying

Function	Description	Example	Output
`copy()`	Shallow copy	`b = a.copy()`	`[same elements]`

List Comprehensions

Powerful syntax for creating lists concisely.

```
# Squares
squares = [x**2 for x in range(5)]
# Even numbers
evens = [x for x in range(10) if x % 2 == 0]
```

Built-in Functions with Lists

Function	Description	Example	Output
`len()`	Length	`len([1,2,3])`	`3`
`sum()`	Sum of numbers	`sum([1,2,3])`	`6`
`max()`	Max element	`max([1,5,3])`	`5`
`min()`	Min element	`min([1,5,3])`	`1`
`sorted()`	Sorted copy	`sorted([3,1])`	`[1,3]`

2.4 Advanced Examples

```
# Flatten nested lists
matrix = [[1,2],[3,4],[5,6]]
flat = [x for row in matrix for x in row]  # [1,2,3,4,5,6]

# Filter using comprehension
nums = [1,2,3,4,5]
even_squares = [x**2 for x in nums if x % 2 == 0]  # [4,16]

# Unique elements
unique = list(set([1,2,2,3,3,4]))
```

---

3. TUPLES IN PYTHON

3.1 Definition

A **tuple** is an **ordered, immutable** collection enclosed in parentheses `()`.
They are faster and used for **fixed data**.

```
t = (10, 20, 30)
```

3.2 Access and Immutability

```
print(t[0])      # 10
# t[1] = 25 ✗ → Error (immutable)
```

3.3 Tuple Methods

Method	Description	Example	Output
`count(x)`	Count occurrences	`(1,2,2,3).count(2)`	2
`index(x)`	Return first index	`(1,2,3,2).index(3)`	2

(Only two methods exist since tuples are immutable.)

3.4 Tuple Operations

Operator	Description	Example	Output
`+`	Concatenation	`(1,2) + (3,4)`	`(1,2,3,4)`
`*`	Repetition	`(1,2) * 2`	`(1,2,1,2)`
`in`	Membership	`3 in (1,2,3)`	`True`
`len()`	Length	`len((1,2,3))`	`3`

3.5 Advanced Examples

```
# Tuple unpacking
a, b, c = (10, 20, 30)

# Nested tuple access
t = (1, (2, 3), 4)
print(t[1][0])  # 2

# Swapping without temp variable
a, b = b, a
```

---

4. Comparison Summary

Feature	String	List	Tuple
Mutable	✗ No	✓ Yes	✗ No
Ordered	✓ Yes	✓ Yes	✓ Yes
Indexed	✓ Yes	✓ Yes	✓ Yes
Duplicates	✓ Yes	✓ Yes	✓ Yes
Methods	Many	Many	Few
Typical Use	Text	Data collection	Fixed data