Python Cheatsheet

1. Python Syntax & Constraints

1.1 Variables & Data Types

Python is dynamically typed, meaning you don't need to declare variable types explicitly. Variables are references to objects stored in memory.

```
x = 10         # Integer
y = 3.14       # Float
name = "Alice" # String
is_valid = True # Boolean
lst = [1, 2, 3] # List
tpl = (4, 5, 6) # Tuple
dct = {"a": 1, "b": 2} # Dictionary
```

Different Ways to Declare Variables:

```
a, b, c = 5, "Hello", 3.14  # Multiple assignment
x = y = z = 10  # Assigning the same value to multiple vari
ables
```

Common Mistakes:

- Assigning multiple variables incorrectly:

  ```
  a, b, c = 1, 2  # ValueError (should match count of valu
  es)
  ```

---

2. Loops & Control Statements

2.1 For Loop (Iterating Over Sequences)

The `for` loop iterates over an iterable object like a list, tuple, or range.

```
for i in range(5):  # range(start, stop, step) is used for iteration
    print(i)  # Outputs: 0, 1, 2, 3, 4
```

2.2 While Loop (Executing Until a Condition is Met)

The `while` loop runs as long as the condition is `True`.

```
x = 0
while x < 5:
    print(x)
    x += 1  # Incrementing to avoid infinite loop
```

2.3 Loop Control Statements

- `break`: Exits the loop early.
- `continue`: Skips the current iteration.

```
for num in range(5):
    if num == 3:
        continue  # Skips 3
    print(num)
```

---

3. Lists (Mutable & Ordered)

Lists are dynamic, mutable, and can hold heterogeneous data types.

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

3.1 List Methods with Explanations

Method	Description	Example	Common Mistake
`append(value)`	Adds an item at the end	`fruits.append("grape")`	Using append for multiple values (use extend instead)

`extend(iterable)`	Adds multiple elements	`fruits.extend(["grape", "mango"])`	Using append instead of extend
`insert(index, value)`	Inserts value at index	`fruits.insert(1, "orange")`	Using an index greater than length
`remove(value)`	Removes first occurrence	`fruits.remove("banana")`	Removing a non-existent value raises error
`pop(index=-1)`	Removes and returns element	`fruits.pop(1)`	Using invalid index
`clear()`	Removes all items	`fruits.clear()`	-
`index(value)`	Finds the index of value	`fruits.index("cherry")`	Searching for missing value raises error
`count(value)`	Counts occurrences	`fruits.count("apple")`	-
`sort()`	Sorts list in place	`numbers.sort()`	Sorting different data types
`reverse()`	Reverses list order	`numbers.reverse()`	-
`copy()`	Copies list	`new_list = fruits.copy()`	Using `=` instead (which creates a reference)

4. Tuples (Immutable & Ordered)

Tuples are similar to lists but immutable (cannot be changed after creation).

```
tpl = (1, 2, 3)
```

4.1 Tuple Methods

Method	Description	Example	Common Mistake
`count(value)`	Counts occurrences	`tpl.count(2)`	-

`index(value)`	Finds index	`tpl.index(3)`	Searching for non-existent value raises error

5. Dictionaries (Key-Value Store)

Dictionaries store data in key-value pairs.

```
dct = {"name": "Alice", "age": 25}
```

5.1 Dictionary Methods with Explanation

Method	Description	Example	Common Mistake
`keys()`	Returns keys	`dct.keys()`	-
`values()`	Returns values	`dct.values()`	-
`items()`	Returns key-value pairs	`dct.items()`	-
`get(key, default)`	Gets value with default	`dct.get("name", "Unknown")`	-
`update(dict)`	Updates dictionary	`dct.update({"age": 26})`	-
`pop(key)`	Removes and returns value	`dct.pop("age")`	Removing a non-existent key
`clear()`	Removes all elements	`dct.clear()`	-

6. Strings (Immutable & Ordered)

Strings are immutable sequences of characters.

```
s = "Hello, World!"
```

6.1 String Methods Explained

Method	Description	Example	Common Mistake
`lower()`	Converts to lowercase	`s.lower()`	-

`upper()`	Converts to uppercase	`s.upper()`	-
`strip()`	Removes whitespace	`s.strip()`	-
`split(delimiter)`	Splits into list	`s.split(",")`	-
`replace(old, new)`	Replaces substring	`s.replace("Hello", "Hi")`	-
`find(value)`	Finds first index	`s.find("World")`	Searching for missing value returns -1
`count(value)`	Counts occurrences	`s.count("o")`	-
`startswith(prefix)`	Checks start	`s.startswith("Hello")`	-
`endswith(suffix)`	Checks end	`s.endswith("!")`	-