Python Data Types Cheatsheet

01. String and String Methods

```
>>> #lower(): Convert to lowercase.
"Data".lower() # "data"

>>> #upper(): Convert to uppercase.
"data".upper() # "DATA"

>>> #strip(): Remove leading/trailing whitespace
" hello ".strip() # "hello"

>>> #lstrip(): Remove leading/trailing whitespace on left
" hello  ".lstrip() # "hello  "

>>> #rstrip(): Remove leading/trailing whitespace on right
" hello  ".rstrip() # " hello"

>>> #.find(): Find index of substring.
"analytics".find("y") # 4

>>> #.startswith(): Check if string starts with a substring.
"report.csv".startswith("report") # True

>>> #endswith(): Check if string ends with a substring.
"report.csv".endswith(".csv") # True

>>> #in: Check if substring exists in string.
"id" in "user_id" # True

>>> #.split(sep): Split string into list
"apple,banana,organge,mango".split(",") # ["apple", "banana"]

>>> #.join(iterable): Join list into string.
"-".join(["2025", "06", "03"]) # "2025-06-03"

>>> #.replace(old, new): Replace substring.
"N/A".replace("N/A", "") # ""

>>> #.title(): Title-case string.
"data science".title() # Data Science

>>> #.capitalize(): Capitalize first letter.
"ricardo kaka".capitalize() # Ricardo kaka

>>> #.isdigit(): Check if all chars are digits.
"123".isdigit() # True

>>> #.isalpha(): Check if all chars are letters.(No space even)
"king".isalpha() # True
```

Python Data Types Cheatsheet

```
>>> #.isalnum(): Check if all chars are alphanumeric.(No Space even)
"error404".isalnum() # True

>>> #.isspace(): Check if string has only spaces.
"    ".isspace() # True

>>> #f-strings: Formatted string literals.
name = "Kohli"
game = "T20"
f"India's run machine is {name} in {game}"

>>> #.format(): Format string using placeholders.
"India's run machine is {} in {}".format("Kohli","T20")

>>> #.split(separator)
fruits = "Mango,Apple,Guava"
fruit_list = fruits.split(",")
fruit_list

>>> fruits_str = ",".join(fruit_list)
fruits_str
```

Strings are immutable, any change in the base string basically creates a new string

```
>>> str1 = "Kaka is the best football player"
print(str1)
print(str1.replace("Kaka","Ronaldo"))
print(str1)

>>> name = "Farhan"
print(name.upper())
print(name.lower())
print(name)

>>>
```

Python Data Types Cheatsheet

02. Operators

```
>>> # 1. Arithmetic Operators
# Operator Precedence PEMDAS
def arithmetic_operators(a, b):
    print("Arithmetic Operators:")
    print(f"{a} + {b} = {a + b}")       # Addition
    print(f"{a} - {b} = {a - b}")       # Subtraction
    print(f"{a} * {b} = {a * b}")       # Multiplication
    print(f"{a} / {b} = {a / b}")       # Division (float)
    print(f"{a} // {b} = {a // b}")     # Floor Division (integer)
    print(f"{a} % {b} = {a % b}")       # Modulus (remainder)
    print(f"{a} ** {b} = {a ** b}")     # Exponentiation

>>> # 2. Comparison Operators
def comparison_operators(x, y):
    print("\nComparison Operators:")
    print(f"{x} == {y}: {x == y}")      # Equal to
    print(f"{x} != {y}: {x != y}")      # Not equal to
    print(f"{x} > {y}: {x > y}")        # Greater than
    print(f"{x} < {y}: {x < y}")        # Less than
    print(f"{x} >= {y}: {x >= y}")      # Greater than or equal to
    print(f"{x} <= {y}: {x <= y}")      # Less than or equal to

>>> def logical_operators(p, q):
    print("\nLogical Operators:")
    print(f"{p} and {q}: {p and q}")    # Logical AND
    print(f"{p} or {q}: {p or q}")      # Logical OR
    print(f"not {p}: {not p}")          # Logical NOT

>>> # 4. Assignment Operators
def assignment_operators(value):
    print("\nAssignment Operators:")
    x = value
    print(f"Initial value: x = {x}")
    x += 5
    print(f"x += 5 -> {x}")
    x -= 2
    print(f"x -= 2 -> {x}")
    x *= 3
    print(f"x *= 3 -> {x}")
    x /= 2
    print(f"x /= 2 -> {x}")
    x //= 2
    print(f"x //= 2 -> {x}")
    x %= 3
    print(f"x %= 3 -> {x}")
    x **= 2
    print(f"x **= 2 -> {x}")
```

Python Data Types Cheatsheet

```
>>> # 5. Bitwise Operators
def bitwise_operators(m, n):
    print("\nBitwise Operators:")
    print(f"{m} & {n} = {m & n}")       # Bitwise AND
    print(f"{m} | {n} = {m | n}")       # Bitwise OR
    print(f"{m} ^ {n} = {m ^ n}")       # Bitwise XOR
    print(f"~{m} = {~m}")               # Bitwise NOT (1's complement)
    print(f"{m} << 1 = {m << 1}")       # Left shift
    print(f"{m} >> 1 = {m >> 1}")       # Right shift

>>> # Sample execution
if __name__ == "__main__":
    arithmetic_operators(10, 3)
    comparison_operators(10, 3)
    logical_operators(True, False)
    assignment_operators(10)
    bitwise_operators(5, 3)

>>>
```

Python Data Types Cheatsheet

03. Operator Predence

```
>>> result = 3 + 4 * 2
print(result)  # 11 -> Multiplication (*) has higher precedence than addition (+)

>>> result = (3 + 4) * 2
print(result)  # 14 -> Parentheses alter precedence; addition happens first

>>> result = 10 - 3 ** 2
print(result)  # 1 -> Exponentiation (**) happens before subtraction (-)

>>> result = 100 / 10 * 5
print(result)  # 50.0 -> / and * have the same precedence, evaluated left to right

>>> result = True or False and False
print(result)  # True -> 'and' has higher precedence than 'or'

>>> result = 3**2**2
print(result) # 81 -> 3**(2**2)
```

All operators have left to rigth precedence, but exponent has right to left precedence.

Python Data Types Cheatsheet

04. If-elif-else

```
>>> #1. Check Missing Values
value = None
if value is None:
    print("Value is null")
else:
    print("Value is not null")

>>> #2. Categorize Age Groups
age = 60
if age<=11:
    print("Kid")
elif age>=12 and age<=18:
    print("Teenager")
elif age>=19 and age<=60:
    print("Adult")
else:
    print("Senior Citizen")

>>> #3. Data Type Checker isinstance(iterable,data_type)
        #isinstance(iterable,datatype) retrurns Boolean only
data1 = [10,20,30]
data2 = {1:"Rahul",2:"Ramesh",3:"Suresh"}
if isinstance(data1,list):
    print("List")
elif isinstance(data1,dictionary):
    print("Dictionary")
else:
    print("Idk bro")

>>> if isinstance(data2,list):
    print("List")
elif isinstance(data2,dict):
    print("Dictionary")
else:
    print("Idk bro")

>>> my_list = [10,20,30,40,50]
print(isinstance(my_list,dict))
print(isinstance(my_list,str))
print(isinstance(my_list,list))

>>> # 4. Check for Outliers
value = 4502         # The number we're checking
mean = 50            # The average of all numbers
threshold = 3        # How far is "too far"? This is our limit.
std_dev = 20         # Standard deviation = how spread out the data is

if abs(value - mean) > threshold * std_dev:
```

```
    print("Outlier detected")
else:
    print("Value is normal")

>>> #5. Detect Empty Dataset
data_set = []   #By default if the data_set is empty its treated as False
data_set_value = [1,2,3,4,5,6,7]
if not data_set:      #If not false(true):
    print("Empty Set")
else:
    print("Not Empty")

if not data_set_value:
    print("Empty Set")
else:
    print("Not Empty")

>>> #6. Model Evaluation Based on Accuracy
accuracy = 0.82

if accuracy >= 0.9:
    print("Excellent Model")
elif accuracy >= 0.75:
    print("Good Model")
elif accuracy >= 0.6:
    print("Average Model")
else:
    print("Poor Model")

>>> #7. Check for NaN (NumPy)
import numpy as np

x = np.nan
#NaN is not equal to anything, not even to itself!

if np.isnan(x):
    print("Value is NaN")
else:
    print("Value is not NaN")

>>> #8. Classify Sentiment Based on Score
score = -0.2

if score > 0:
    print("Positive Sentiment")
elif score < 0:
    print("Negative Sentiment")
else:
    print("Neutral Sentiment")

>>> #9. Check Column Presence in DataFrame
```

Python Data Types Cheatsheet

```
import pandas as pd

df = pd.DataFrame({'name': ['Alice'], 'age': [25]})
print(df)

if 'salary' in df.columns:
    print("Salary column exists")
else:
    print("Salary column missing")

>>> #10. Determine Leap Year
year = 2024

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print("Leap year")
else:
    print("Not a leap year")

>>>
```

Python Data Types Cheatsheet

05. Match Case

```
>>> # HTTP Server Error Code
def http_response_code(code):
    match code:
        case 500 : print("Internal Server Error")
        case 404 : print("Page Not Found")
        case 400 : print("Bad Request")
        case _ : print("Unknown Error")
http_response_code(500)

>>> #1. Match column data type (from pandas)
def describe_column(dtype):
    match dtype:
        case 'int64':
            return "Integer column"
        case 'float64':
            return "Floating point column"
        case 'object':
            return "Categorical/Text column"
        case _:
            return "Other data type"

print(describe_column('object'))  # Output: Categorical/Text column

>>> #2. Handle ML model names
def handle_ml_model(model_name):
    match model_name.lower():
        case "kmeans" :
            return "Clustering Model"
        case "linear regression" :
            return "Regressin Model"
        case "decision tree":
            return "Classification Model"
        case _:
            return "Unknown Model"
print(handle_ml_model("Decision tree"))

>>> #3. Analysing number of columns
def no_of_cols(col_count):
    match col_count:
        case 0 :
            return "Empty Dataset"
        case 1:
            return "Only Index Col"
        case 2:
            return "Two Columns"
        case _ if col_count>3 and col_count<10:
            return "Normal Count"
        case _:
```

```
            return "Out of Scope"
print(no_of_cols(2))
print(no_of_cols(5))
print(no_of_cols(100))

>>> def file_type(filename):
    match filename.split(".")[-1]:
        case "csv" :
            return"Comma-Separated Value"
        case "pdf" :
            return "PDF File"
        case "txt" :
            return "Text File"
        case _:
            return "Unknown File Type"

filename="diwali_sales.csv"
file_type(filename)

>>> #4. Matching Datastructures
point = (2,0)
def locate_point(point):
    match point:
        case (0,0) :
            return "Origin"
        case (x,0) :
            return "On X-Axis at {}".format(x)
        case (0,y):
            return "On Y-Axis at {}".format(y)
        case _:
            return "In the open plane"
locate_point(point)
```

Python Data Types Cheatsheet

06. f-Strings and format

.format()

```
>>> name = "Alice"
age = 25
print("My name is {} and I am {} years old.".format(name, age))
```

- Positional Argument in .format()

```
>>> print("My age is {1} and my name is {0}".format(name, age))
```

- Keyword Argument in .format()

```
>>> print("My age is {age} years and My name is {name}".format(age=35,name="Kohli"))
```

f-string

```
>>> name = "Virat"
age = 29
print(f"India's run machine : {name} is {age} years old")

>>> # Expressions Inside f-Strings
a = 5
b = 10
print(f"Sum of {a} and {b} is {a + b}")
# Output: Sum of 5 and 10 is 15

>>> #Formatting Numbers
pi = 3.14159
print(f"Value of pi rounded to 2 decimal places is: {pi:.2f}")

>>> #Padding and Alignment
print(f"{'Python':<10}") # Left-align
print(f"{'Python':>10}") # Right-align
print(f"{'Python':^10}") # Center-align
#:<10 → The < symbol means left-align the text within a total width of 10 characters
print(f"{'Python':<1}")

>>>
```

Python Data Types Cheatsheet

08. List and List Methods

```
>>> # Empty list
my_list = []
print(my_list)
# List with elements
numbers = [1, 2, 3, 4, 5]
print(numbers)

>>> """ Why it works:
In Python, empty sequences (like [], '', {}) are treated as False when used in a condition.

So, not my_list becomes True if the list is empty. """
my_list = []
if not my_list:
    print("List is empty")
else:
    print("List is not empty")
```

- string.split("separator") : Converts str into list

```
>>> str1 = "Apple,Mano,Banana"
list1 = str1.split(",")
print(list1)
```

- "separator".join(list) : Converts list into str

```
>>> str2 = ",".join(list1)
str2

>>> # Mixed data types
mixed_list = [1, "Hello", 3.14, True,3j+2]
mixed_list
```

List Methods

```
>>> my_list = [10,"hello"]

>>> #append(x) : Adds an element x to the end of the list.
my_list.append(20)
print(my_list)

>>> #extend(iterable) : Extends the list by appending all elements from an iterable.\
ext_list = [30,"Jo"]
my_list.extend(ext_list)
print(my_list)

>>> #insert(index, x) : Inserts x at the specified index .
print(my_list[2])
my_list.insert(2,200)
```

Python Data Types Cheatsheet

```
print(my_list)

>>> #remove(x) : Removes the first occurrence of x in the list.
my_list.remove(30)
print(my_list)

>>> #pop([index]) : Removes and returns the element at index (last element if index is not provided).
print(my_list[2])
popped_elem = my_list.pop(2)
print(my_list)
print(popped_elem)

>>> # index(x) : Returns the index of the first occurrence of x .
print(my_list.index('hello'))

>>> #count(x) : Returns the number of times x appears in the list.
my_list.count('Jo')

>>> #sort() : Sorts the list in ascending order.
#my_list.sort()  will show error because the list has both str and int vals

>>> #reverse() : Reverses the order of the list.
print(my_list)
my_list.reverse()
print(my_list)

>>> #copy() : Returns a shallow copy of the list.
print(my_list)
copy_list = my_list.copy()
print(copy_list)

>>> my_list[1] = "Yo"
print(my_list)
print(copy_list)

>>> print(copy_list)
copy_list[2] = "Hon"
print(my_list)
print(copy_list)

| Copy Type    | Affects Other List? | When?                                 |
| ------------ | ------------------- | ------------------------------------- |
| `.copy()`    | □ No                | If list contains only simple values   |
| `.copy()`    | □ Yes               | If list contains nested/mutable values |
| `deepcopy()` | □ Never             | Completely safe, but slower           |

>>> fruits = ['apple', 'watermelom','banana','peach', 'cherry', 'orange']
fruits.sort()
print(fruits)

**Difference between sort() and sorted()**
```

Python Data Types Cheatsheet

Feature	`sort()`	`sorted()`
Applicable to	Lists only	Any iterable (list, tuple, set, dict, etc.)
Return value	Returns `None`	Returns a **new sorted list**
In-place modification	□ Yes (modifies the original list)	□ No (original data remains unchanged)
Original data	Changes	Unchanged
Usage syntax	`list.sort()`	`sorted(iterable)`
Flexibility	Less flexible	More flexible (can work on any iterable)
Performance	Slightly faster for lists	Slightly slower due to creating a new list

```
>>> list1 = [10,2,3,5,22,-9,-10]
print(list1)  # Sorts the original list

>>> list2 = [-9.100,33,12,-8.12]
new_list2 = sorted(list2)
print(list2)  # Original list remains unchanged
print(new_list2)  # A new sorted list is created

>>> # Concatenation of list
list1 = [10,12,3,4,9]
list2 = [5,32,97,-4]
list3 = list1+list2
list3.sort()
print(list3)
print(list2)  # Original List remains unchanged
print(list1)  # Original List remains unchanged
```

Membership of Element in List

```
>>> list1 = [10,12,3,4,9]
if 3 in list1:          #(3 in list1) returns a boolean value
    print("yes")
else:
    print("no")

>>> crew = "jack-john-salman-rashid"
(crew.title()).split("-")

>>> players = "dhoni-rohit-kohli-jadeja-shami"
pl_list = players.split("-")
ans = [i.title() for i in pl_list]
print(ans)

>>> players = "dhoni-rohit-kohli-jadeja-shami"
pl_list = players.split("-")
ans = []
for i in pl_list:
    ans.append(i.title())
print(ans)
```

Python Data Types Cheatsheet

10. Tuples and Methods

Tuples are same as Lists but the only difference is they are immuatable. Tuples are faster than lists

```
>>> # Empty Tuple
tup = ()
tup

>>> # Single Element Tuple
tup = (1)
print(type(tup))
true_tup = (1,)   # , is necessary
print(type(true_tup))

>>> # Tuple
tup = (2,4,5,23)
tup

>>> # Mixed Tuple
tup = (1,2,5+4j,True,5.12,"Yo")
tup

>>> # Check if the tuple is empty
my_tup = ()
if not my_tup:
    print("Empty")
else:
    print("Not Empty")
```

Tuple Methods

```
>>> newtup = (10,2,30,"Hi",99,-12,10,2,2,2)
newtup

>>> newtup.count(2)

>>> newtup.index(2)

>>> len(newtup)

>>> # Tuple Packing
data = 10,"hi",True  # same as data = (10,"hi",True)
print(data)

>>> # Tuple Unpacking
num,msg,status = data
print(num)
print(msg)
print(status)
```

Python Data Types Cheatsheet

```
>>> # Memebership Testing
10 in newtup

>>> # Concatenation
t1 = (1, 2)
t2 = (3, 4)
t3 = t1 + t2  # Output: (1, 2, 3, 4)
t3

>>> # Repeatation
t1 = (1, 2)
t1 * 3  # Output: (1, 2, 1, 2, 1, 2)

>>> #Zip
names = ['Alice', 'Bob']
scores = [85, 90]
card = list(zip(names,scores))
card #List of Tuples

>>> card_tup = tuple(zip(names,scores))
card_tup #Nested Tuple

Accessing Tuple Elements

>>> tuple1 = (10,20,10,2,23,10)
tuple1[2]

>>> tuple1[::-1]
```

Python Data Types Cheatsheet

11. Sets and Methods

Unordered, Mutable and Unique. Set always stores values in sorted manner

```
>>> # Empty Set
empty_set = set()
empty_set

>>> # Number set
nos = {1,2,3,4,5}
nos

>>> # Mixed Set
mixed_set = {1,2,True,5+4j,7.123,"YO"}
mixed_set

>>> # Converting list into set -- be observant about uniqueness
new_set1 = set([1,2,2,3,4,5,5,True,True])
new_set1  #True is treated as the number 1 in Python, because True ==1

>>> new_set2 = set([2,2,"Yo","Yo","Yo",False,False,2,2,True,True])
new_set2

>>> final_set = {1,2,3,4}
final_set

>>> final_set.add(22) #Add one element
final_set

>>> final_set.update([10,20,30,40,50]) # Add multiple elements
final_set

>>> final_set.remove(40)
final_set

>>> #final_set.remove(40)  #raises error if not found
final_set.discard(50)
final_set.discard(50) # Doesnt raises error if not found

>>> pop_elem = final_set.pop()  #returns and removes random elements
final_set

>>> doomsday = final_set.copy() #SHallow Copy
doomsday

>>> final_set.clear()

>>> final_set

>>> doomsday
```

Python Data Types Cheatsheet

Set Operations

```
>>> #1. Union
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)              # {1, 2, 3, 4, 5}
print(a.union(b))         # {1, 2, 3, 4, 5}

>>> #2. Intersection (& or .intersection())
print(a & b)              # {3}
print(a.intersection(b))   # {3}

>>> # 3. Difference (- or .difference())
print(a - b)              # {1, 2}
print(a.difference(b))     # {1, 2}

>>> #4. Symmetric Difference (^ or .symmetric_difference()) : Items in either set, but not in both.
print(a ^ b)                    # {1, 2, 4, 5}
print(a.symmetric_difference(b))  # {1, 2, 4, 5}

>>> #Subset and Superset
a = {1, 2}
b = {1, 2, 3, 4}

print(a <= b)  # True  (a is a subset of b)
print(b >= a)  # True  (b is a superset of a)

>>> #6. Disjoint Sets (.isdisjoint()): Check if two sets have no elements in common
x = {1, 2}
y = {3, 4}

print(x.isdisjoint(y))  # True
```

Python Data Types Cheatsheet

12. Dictionary and Methods

Python dictionaries are key indexed. ex: nameOfDict[key] will work

```
>>> mydict = {
    1: "Mohammad Areeb Ahmad",
    2: "Ricardo Kaka",
    3: "Lionel Messi",
    4: "Ronaldo"
}
mydict

>>> # Check if dictionary is empty
if not mydict:
    print("Empty")
else:
    print("True")

>>> record = {
    'name': 'Alice',
    'age': 30,
    'profession': 'Data Scientist',
    'skills': ['Python', 'SQL', 'Pandas']
}

>>> # 1. get() : Safely access a key — no error if key doesn't exist.
print(record.get('age'))              # 30
print(record.get('gender', 'N/A'))    # 'N/A'
print(record.get('degree'))           # None

>>> #2. items() : Loop over keys and values.
for key, value in record.items():
    print(f"{key} : {value}")

>>> #3. keys()
print(record.keys())
print(list(record.keys()))

>>> #4. values()
print(record.values())
print(list(record.values()))

>>> #7. setdefault() : Set a default if key doesn't exist.
record.setdefault("City","Delhi")
record

>>> #8.fromkeys() : Make a new dict from a list of keys.
fields = ['name', 'age', 'gender']
empty_profile = dict.fromkeys(fields, None)
print(empty_profile)
```

Python Data Types Cheatsheet

```
# {'name': None, 'age': None, 'gender': None}

>>> record_copy = record.copy()
print(record_copy)

>>> #10. .pop() :Removes the specified key from the dictionary and returns its value.
record = {'name': 'Alice', 'age': 30, 'profession': 'Data Scientist'}

value = record.pop('age')
print(value)     # 30
print(record)    # {'name': 'Alice', 'profession': 'Data Scientist'}

# If key doesn't exist:
record.pop('city', 'Not Found')   # returns 'Not Found'
#record.pop('gender')              #  raises KeyError

>>> #11. .popitem() : Removes and returns the last inserted (key, value) pair as a tuple.
record = {
   'name': 'Alice',
   'age': 30,
   'profession': 'Data Scientist'
}

item = record.popitem()
print(item)      # ('profession', 'Data Scientist')
print(record)    # {'name': 'Alice', 'age': 30}

>>> # If dict is empty:
empty = {}
#empty.popitem()  #  KeyError

>>> mydict1 = {
   1: "Mohammad Areeb Ahmad",
   2: "Ricardo Kaka",
   3: "Lionel Messi",
   4: "Ronaldo"
}
mydict1

>>> mydict1[1]

>>> mydict1[3]

>>> mydict1.update({4:"Pele"})

>>> mydict1

>>> mydict1.update({2:"Kafu",5:"Zidane",6:"Marcelo"})
mydict1

>>>
```

Python Data Types Cheatsheet

>>>