Python

Basic Python syntax - variables, comments, input/output
Data types - numbers, strings, booleans with operations
Control flow - if-else statements, loops, conditions
Functions - defining, parameters, lambda functions
Data structures - lists, tuples, dictionaries, sets
File operations - reading/writing text, CSV, and JSON
Error handling - try-except blocks
Classes and OOP - object creation, inheritance, properties
Modules and packages - importing, creating your own
Common libraries - math, datetime, regex, HTTP requests

Python Cheatsheet

Table of Contents

Basics

Hello World

```
print("Hello, World!")
```

Variables and Assignment

```
x = 5           # Integer
name = "Python"    # String
is_active = True   # Boolean
```

Comments

```
# This is a single-line comment

"""
This is a
multi-line comment
"""
```

Input and Output

```
name = input("Enter your name: ")  # Get user input
print(f"Hello, {name}!")         # f-string (Python 3.6+)
print("Hello, {}!".format(name))   # .format() method
print("Hello, " + name + "!")     # String concatenation
```

Importing Modules

```
import math               # Import whole module
from datetime import date    # Import specific items
import numpy as np         # Import with alias
```

Data Types

Numbers

```
x = 5       # Integer
y = 3.14    # Float
z = 1 + 2j  # Complex number

# Operations
sum = x + y
diff = x - y
product = x * y
```

```
quotient = x / y      # Float division: 5/2 = 2.5
floor_div = x // y    # Integer division: 5//2 = 2
remainder = x % y     # Modulo: 5%2 = 1
power = x ** 2        # Exponent: 5^2 = 25
```

Strings

```
s1 = 'Single quotes'
s2 = "Double quotes"
s3 = '''Multi-line
string'''

# String operations
length = len(s1)            # String length
concat = s1 + " " + s2        # Concatenation
repeat = "abc" * 3          # "abcabcabc"
char = s1[0]                # First character: 'S'
substring = s1[0:6]          # Slice: 'Single'
uppercase = s1.upper()        # Convert to uppercase
lowercase = s1.lower()        # Convert to lowercase
replace = s1.replace('S', 'R') # Replace: 'Ringle quotes'
split_str = "a,b,c".split(',') # Split: ['a', 'b', 'c']
joined = "-".join(['a', 'b'])  # Join: "a-b"
strip_str = " text ".strip()   # Remove whitespace: "text"
```

Booleans

```
a = True
b = False

# Boolean operations
and_op = a and b    # False
or_op = a or b      # True
not_op = not a      # False

# Comparison operators
eq = (5 == 5)       # Equal: True
ne = (5 != 10)      # Not equal: True
gt = (5 > 3)        # Greater than: True
lt = (5 < 10)       # Less than: True
```

```
ge = (5 >= 5)      # Greater or equal: True
le = (5 <= 10)     # Less or equal: True
```

Type Conversion

```
str_to_int = int("42")          # String to integer
int_to_str = str(42)            # Integer to string
str_to_float = float("3.14")    # String to float
bool_value = bool(0)            # Convert to boolean (0 is False)
```

Control Flow

If-Else Statements

```
x = 10

if x > 10:
    print("x is greater than 10")
elif x < 10:
    print("x is less than 10")
else:
    print("x is equal to 10")
```

Ternary Operator

```
age = 20
status = "adult" if age >= 18 else "minor"
```

For Loops

```
# Loop through a range
for i in range(5):  # 0, 1, 2, 3, 4
    print(i)

# Loop through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
# Loop with enumerate (index and value)
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
```

While Loops

```
count = 0
while count < 5:
    print(count)
    count += 1

# Break and continue
while True:
    if count >= 10:
        break  # Exit the loop
    if count % 2 == 0:
        count += 1
        continue # Skip to next iteration
    print(count)
    count += 1
```

Functions

Basic Function

```
def greet(name):
    """This is a docstring that explains the function."""
    return f"Hello, {name}!"

# Calling the function
message = greet("Alice")
```

Arguments and Parameters

```
# Default parameter values
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"
```

```
# Positional arguments
result1 = greet("Bob")       # "Hello, Bob!"
result2 = greet("Bob", "Hi") # "Hi, Bob!"

# Keyword arguments
result3 = greet(greeting="Howdy", name="Charlie") # "Howdy, Charlie!"

# Variable number of arguments
def add_all(*args):           # Tuple of arguments
    return sum(args)

sum_result = add_all(1, 2, 3, 4) # 10

# Variable keyword arguments
def print_info(**kwargs):       # Dictionary of keyword arguments
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Alice", age=30, city="New York")
```

Lambda Functions (Anonymous Functions)

```
square = lambda x: x**2
print(square(5))  # 25

# With map()
numbers = [1, 2, 3, 4]
squares = list(map(lambda x: x**2, numbers))  # [1, 4, 9, 16]

# With filter()
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))  # [2, 4]
```

Data Structures

Lists

```
# Creating lists
fruits = ["apple", "banana", "cherry"]
mixed = [1, "hello", True, 3.14]
```

```
nested = [1, [2, 3], 4]

# Accessing elements
first = fruits[0]        # "apple"
last = fruits[-1]        # "cherry"
sublist = fruits[0:2]    # ["apple", "banana"]

# Modifying lists
fruits.append("orange")   # Add to end
fruits.insert(1, "mango") # Insert at index 1
fruits.remove("banana")   # Remove by value
popped = fruits.pop()     # Remove and return last item
popped_index = fruits.pop(0)  # Remove by index
fruits[0] = "pear"        # Replace element
fruits.sort()             # Sort in-place
fruits.reverse()          # Reverse in-place
sorted_fruits = sorted(fruits)  # Return sorted copy
length = len(fruits)      # Number of items

# List comprehensions
squares = [x**2 for x in range(10)]
even_squares = [x**2 for x in range(10) if x % 2 == 0]
```

Tuples (Immutable)

```
# Creating tuples
coordinates = (10, 20)
single_item = (1,)       # Comma is required for single item
mixed = (1, "hello", True)

# Accessing elements (same as lists)
x = coordinates[0]       # 10

# Operations
combined = coordinates + (30, 40)  # (10, 20, 30, 40)
repeated = coordinates * 2         # (10, 20, 10, 20)

# Unpacking
x, y = coordinates       # x = 10, y = 20
```

Dictionaries

```
# Creating dictionaries
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

# Accessing elements
name = person["name"]       # "Alice"
age = person.get("age")     # 30
age = person.get("height", 0) # Returns 0 if key doesn't exist

# Modifying dictionaries
person["email"] = "alice@example.com"  # Add new key-value pair
person["age"] = 31                 # Modify existing value
person.update({"phone": "123-456-7890", "age": 32})  # Update multiple
del person["city"]                 # Remove key-value pair
popped = person.pop("age")          # Remove and return value

# Dictionary operations
keys = person.keys()         # dict_keys(['name', 'email', 'phone'])
values = person.values()     # dict_values(['Alice', 'alice@example.com', '123-456-7890'])
items = person.items()       # dict_items([('name', 'Alice'), ...])

# Dictionary comprehension
squares = {x: x**2 for x in range(6)}  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Sets

```
# Creating sets
fruits = {"apple", "banana", "cherry"}
numbers = set([1, 2, 2, 3, 4, 4])  # Creates {1, 2, 3, 4}

# Set operations
fruits.add("orange")         # Add an element
fruits.remove("banana")      # Remove (raises error if not found)
fruits.discard("banana")     # Remove (no error if not found)
popped = fruits.pop()        # Remove and return an arbitrary element
length = len(fruits)         # Number of elements
```

```
# Set operations
A = {1, 2, 3}
B = {3, 4, 5}
union = A | B              # {1, 2, 3, 4, 5}
intersection = A & B       # {3}
difference = A - B         # {1, 2}
symmetric_diff = A ^ B     # {1, 2, 4, 5}

# Set comprehension
even_set = {x for x in range(10) if x % 2 == 0}  # {0, 2, 4, 6, 8}
```

File Operations

Reading Files

```
# Basic file reading
file = open("example.txt", "r")
content = file.read()      # Read entire file
file.close()

# Using with statement (auto-closes file)
with open("example.txt", "r") as file:
    content = file.read()  # Read entire file

with open("example.txt", "r") as file:
    lines = file.readlines()  # Read as list of lines

with open("example.txt", "r") as file:
    for line in file:      # Read line by line
        print(line.strip())  # strip() removes trailing newline
```

Writing Files

```
# Write to file (overwrites existing content)
with open("output.txt", "w") as file:
    file.write("Hello, World!\n")
    file.write("Another line.")
```

```
# Append to file
with open("output.txt", "a") as file:
    file.write("\nAppended line.")

# Write multiple lines
lines = ["line 1", "line 2", "line 3"]
with open("output.txt", "w") as file:
    file.writelines(f"{line}\n" for line in lines)
```

Working with CSV Files

```
import csv

# Reading CSV
with open("data.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)  # row is a list of values

# Reading CSV as dictionaries
with open("data.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row)  # row is a dictionary

# Writing CSV
data = [["Name", "Age"], ["Alice", 30], ["Bob", 25]]
with open("output.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(data)

# Writing CSV from dictionaries
data = [
    {"Name": "Alice", "Age": 30},
    {"Name": "Bob", "Age": 25}
]
with open("output.csv", "w", newline="") as file:
    fieldnames = ["Name", "Age"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
```

```
    writer.writerows(data)
```

Working with JSON Files

```
import json

# Reading JSON
with open("data.json", "r") as file:
    data = json.load(file)  # data is a Python object (dict, list, etc.)

# Writing JSON
data = {"name": "Alice", "age": 30, "cities": ["New York", "London"]}
with open("output.json", "w") as file:
    json.dump(data, file, indent=4)  # indent for pretty printing

# Converting Python objects to JSON strings
json_string = json.dumps(data, indent=4)

# Converting JSON strings to Python objects
data = json.loads('{"name": "Alice", "age": 30}')
```

Error Handling

Try-Except Blocks

```
# Basic try-except
try:
    x = 10 / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError:
    print("Cannot divide by zero!")

# Multiple exception types
try:
    num = int("abc")  # This will raise a ValueError
except ValueError:
    print("Invalid conversion!")
except ZeroDivisionError:
    print("Division by zero!")
```

```
# Catching any exception
try:
    x = 10 / 0
except Exception as e:
    print(f"An error occurred: {e}")

# Else and finally clauses
try:
    x = 10 / 2
except ZeroDivisionError:
    print("Division by zero!")
else:
    print("No error occurred!")  # Executes if no exception
finally:
    print("This always executes!")  # Always executes
```

Raising Exceptions

```
def validate_age(age):
    If age < 0:
        raise ValueError("Age cannot be negative")
    If age > 150:
        raise ValueError("Age is too high")
    return age

# Raising a custom message
If not isinstance(name, str):
    raise TypeError("Name must be a string")
```

Classes and OOP

Basic Class Definition

```
class Person:
    # Class variable (shared by all instances)
    species = "Homo sapiens"

    # Constructor (initializer)
    def __init__(self, name, age):
```

```
        # Instance variables (unique to each instance)
        self.name = name
        self.age = age

    # Instance method
    def greet(self):
        return f"Hello, my name is {self.name}"

    # Method with parameters
    def celebrate_birthday(self):
        self.age += 1
        return f"Happy {self.age}th birthday, {self.name}!"

    # String representation
    def __str__(self):
        return f"Person(name='{self.name}', age={self.age})"

    # Representation (for debugging)
    def __repr__(self):
        return f"Person('{self.name}', {self.age})"

# Creating instances
alice = Person("Alice", 30)
bob = Person("Bob", 25)

# Accessing attributes and methods
print(alice.name)          # "Alice"
print(alice.greet())       # "Hello, my name is Alice"
print(alice.celebrate_birthday())  # "Happy 31th birthday, Alice!"
print(alice)               # Calls __str__
```

Inheritance

```
class Student(Person):
    def __init__(self, name, age, student_id):
        # Call parent class constructor
        super().__init__(name, age)
        self.student_id = student_id

    # Override parent's method
```

```
    def greet(self):
        return f"{super().greet()} and I'm a student"

    # New method
    def study(self, subject):
        return f"{self.name} is studying {subject}"

# Create a Student instance
charlie = Student("Charlie", 20, "S12345")
print(charlie.greet())  # "Hello, my name is Charlie and I'm a student"
print(charlie.study("Python"))  # "Charlie is studying Python"
```

Properties and Private Attributes

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self._balance = balance  # Convention for "private" attribute

    @property
    def balance(self):
        """Get the current balance."""
        return self._balance

    @balance.setter
    def balance(self, value):
        """Set the balance with validation."""
        if value < 0:
            raise ValueError("Balance cannot be negative")
        self._balance = value

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount must be positive")
        self._balance += amount
        return self._balance

    def withdraw(self, amount):
        if amount <= 0:
            raise ValueError("Withdrawal amount must be positive")
```

```
        if amount > self._balance:
            raise ValueError("Insufficient funds")
        self._balance -= amount
        return self._balance

# Using the class
account = BankAccount("Alice", 1000)
print(account.balance)  # 1000 (uses the @property getter)
account.balance = 2000  # Uses the @property setter
print(account.deposit(500))  # 2500
print(account.withdraw(1000))  # 1500
```

Static and Class Methods

```
class MathUtils:
    # Class variable
    PI = 3.14159

    # Static method (doesn't use class or instance)
    @staticmethod
    def add(a, b):
        return a + b

    # Class method (uses the class)
    @classmethod
    def circle_area(cls, radius):
        return cls.PI * radius ** 2

# Using static and class methods
print(MathUtils.add(5, 3))        # 8
print(MathUtils.circle_area(5))   # 78.53975
```

Modules and Packages

Creating a Module

```
# mymodule.py
def greet(name):
    return f"Hello, {name}!"
```

```
def add(a, b):
    return a + b

PI = 3.14159

if __name__ == "__main__":
    # This code runs when the module is executed directly
    print(greet("World"))
```

Importing from a Module

```
# Import specific items
from mymodule import greet, PI
print(greet("Alice"))  # "Hello, Alice!"
print(PI)              # 3.14159

# Import everything
from mymodule import *

# Import with alias
import mymodule as mm
print(mm.add(5, 3))    # 8
```

Creating a Package

```
mypackage/
    __init__.py
    module1.py
    module2.py
    subpackage/
        __init__.py
        module3.py
```

Importing from a Package

```
# Import a module from a package
import mypackage.module1
```

```
# Import specific functions
from mypackage.module1 import function1
from mypackage.subpackage.module3 import function2

# Import with aliases
import mypackage.module2 as m2
```

Common Libraries

Math and Statistics

```
import math
import statistics

# Math functions
x = math.sqrt(16)      # Square root: 4.0
y = math.pow(2, 3)     # Power: 8.0
z = math.pi            # Pi: 3.141592...
a = math.floor(3.7)    # Floor: 3
b = math.ceil(3.2)     # Ceiling: 4
c = math.sin(math.pi/2) # Sine: 1.0

# Statistics
data = [1, 2, 3, 4, 5]
mean = statistics.mean(data)   # 3.0
median = statistics.median(data)  # 3.0
stdev = statistics.stdev(data)   # Standard deviation
```

Date and Time

```
from datetime import datetime, date, time, timedelta

# Current date and time
now = datetime.now()     # Current local datetime
today = date.today()     # Current local date

# Creating dates
specific_date = date(2023, 9, 15)  # Year, month, day
specific_time = time(13, 30, 15)   # Hour, minute, second
```

```
specific_dt = datetime(2023, 9, 15, 13, 30, 15)  # Combine

# Formatting
formatted = now.strftime("%Y-%m-%d %H:%M:%S")  # "2023-09-15 13:30:15"

# Parsing
parsed = datetime.strptime("2023-09-15", "%Y-%m-%d")

# Time differences
one_week = timedelta(days=7)
next_week = now + one_week
time_diff = next_week - now  # timedelta object
```

Random

```
import random

# Random number generation
rand_int = random.randint(1, 10)    # Random int between 1 and 10 (inclusive)
rand_float = random.random()        # Random float between 0 and 1
rand_range = random.uniform(1, 10)  # Random float between 1 and 10

# Random selections
choices = ["rock", "paper", "scissors"]
rand_choice = random.choice(choices)   # Select one random item
rand_sample = random.sample(choices, 2) # Select multiple without replacement
random.shuffle(choices)             # Shuffle in-place
```

Regular Expressions

```
import re

text = "The quick brown fox jumps over the lazy dog."

# Pattern matching
match = re.search(r"fox", text)  # Find first match
if match:
    print(match.group())        # "fox"
    print(match.start())        # Start index
    print(match.end())          # End index
```

```
# Find all matches
matches = re.findall(r"\w{4,}", text)  # All words with 4+ letters
# ['quick', 'brown', 'jumps', 'over', 'lazy']

# Replace
new_text = re.sub(r"fox", "cat", text)
# "The quick brown cat jumps over the lazy dog."

# Split
parts = re.split(r"\s+", text)  # Split by whitespace

# Compile pattern for reuse
pattern = re.compile(r"\w+")
words = pattern.findall(text)
```

JSON

```
import json

# Convert Python object to JSON string
data = {
   "name": "Alice",
   "age": 30,
   "is_student": False,
   "courses": ["Python", "Data Science"],
   "address": {
      "city": "New York",
      "zip": "10001"
   }
}
json_str = json.dumps(data, indent=4)

# Parse JSON string
parsed_data = json.loads(json_str)
```

HTTP Requests

```
import requests
```

```
# GET request
response = requests.get("https://api.example.com/data")
data = response.json()  # Parse JSON response
status = response.status_code  # 200 for success

# POST request
payload = {"name": "Alice", "email": "alice@example.com"}
response = requests.post("https://api.example.com/users", json=payload)

# Headers
headers = {"Authorization": "Bearer token123"}
response = requests.get("https://api.example.com/protected", headers=headers)

# Query parameters
params = {"q": "python", "limit": 10}
response = requests.get("https://api.example.com/search", params=params)

# Session for multiple requests
session = requests.Session()
session.headers.update({"Authorization": "Bearer token123"})
response1 = session.get("https://api.example.com/resource1")
response2 = session.get("https://api.example.com/resource2")
```

Virtual Environments

```
# Create a virtual environment
python -m venv myenv

# Activate virtual environment (Windows)
myenv\Scripts\activate

# Activate virtual environment (macOS/Linux)
source myenv/bin/activate

# Install packages
pip install package_name
pip install -r requirements.txt

# Create requirements.txt
pip freeze > requirements.txt
```

```
# Deactivate virtual environment
deactivate
```