🔍 INTRODUCTION OF PYTHON

🐍 What is Python?

Python ek **high-level, interpreted** programming language hai. Ye simple aur readable syntax ki wajah se beginners ke liye best hai.

Key Features:

- Easy to learn
- Cross-platform (Windows, Mac, Linux)
- Multiple paradigms: Object-Oriented, Functional, Procedural
- Rich Libraries: NumPy, Pandas, OpenCV, TensorFlow, etc.

🤖 Python in Agentic AI (Autonomous AI Agents)

Agentic AI wo systems hote hain jo:

1. **Perceive** (mahsoos karte hain input)
2. **Reason** (soch kar faislay lete hain)
3. **Act** (koi kaam anjaam detay hain)

Python Tools for Agentic AI:

- **LangChain** – LLM workflows
- **Auto-GPT / CrewAI / AutoGen** – autonomous decision making
- **OpenAI APIs** – LLM integration
- **NLP & Reinforcement Learning** – learning and adaptation

🛠️ Practical Applications of Python

Field	Use
Data Science	Data cleaning, ML, Visuals
AI / ML	Predictive models
Web Dev	Flask, Django
Robotics	Control systems
NLP	Chatbots, Translation
Cybersecurity	Ethical hacking
Automation	Scripts for repetitive tasks

⚙️ Code Execution Continuum (Behind the Scenes)

1. **Code Writing**: You write Python code (`hello.py`)
2. **Interpretation**: Python interpreter reads it
3. **Bytecode**: Internally converted to `.pyc` (bytecode)
4. **Runtime**: Python Virtual Machine (PVM) executes it
5. **Output**: Final result is displayed

🧠 Part 2: MCQs for Practice

📘 Python Basics

1. **Who created Python?**
   A) Dennis Ritchie
   B) James Gosling
   C) Guido van Rossum
   D) Bjarne Stroustrup
   ✅ Correct: **C**

2. **Which of the following is not a Python paradigm?**
   A) Procedural
   B) OOP
   C) Declarative
   D) Functional
   ✅ Correct: **C**

3. **What does PVM stand for?**
   A) Python Visual Manager
   B) Python Version Maker
   C) Python Virtual Machine
   D) Python Variable Modifier
   ✅ Correct: **C**

📙 Agentic AI & Applications

4. **LangChain is mainly used in Python for...?**
   A) Data Mining
   B) Chain-of-thought workflows in AI
   C) Robotics
   D) UI Design
   ✅ Correct: **B**

5. **Which Python library is best for computer vision?**
   A) NLTK
   B) Pandas
   C) OpenCV
   D) Matplotlib
   ✅ Correct: **C**

6. **Which field uses Python for penetration testing?**
   A) Robotics
   B) Web Development
   C) Cybersecurity
   D) NLP
   ✅ Correct: **C**

💻 Part 3: Code Examples for Understanding

✅ Example 1: Simple Python Agent (Mock)

python
CopyEdit

```
def perceive():
    print("👀 Perceiving environment...")

def reason():
    print("🧠 Thinking and planning...")

def act():
    print("🤖 Taking action!")

def autonomous_agent():
    perceive()
    reason()
    act()

autonomous_agent()
```

Output:

CopyEdit

```
👀 Perceiving environment...
🧠 Thinking and planning...
🤖 Taking action!
```


🔥 Introduction to Python Bytecode

Jab hum Python ka code likhtay hain, to wo directly machine pe run nahi hota. Python pehle us code ko **Bytecode** mein convert karti hai – ye ek intermediate format hota hai jo sirf Python interpreter samajhta hai.

🛠️ Python Code Compilation Process

Step	Tashreeh (وضاحت)
Lexical Analysis	Code ko chhoti chhoti units (tokens) mein torna, jaise name, =, print
Syntax Analysis	Check karna ke code ka structure sahi hai ya nahi (indentation, brackets etc.)
Semantic Analysis	Yeh dekhna ke code ka logic aur context theek hai (e.g., undefined variable to nahi)
Bytecode Generation	Python compiler isay bytecode mein badal deta hai jo interpreter samajhta hai

---

🗒️ What is Python Bytecode?

Python bytecode ek **platform-independent** intermediate code hota hai. Isay Python interpreter hi samajhta hai – yeh machine code nahi hota.

Jab hum koi module import kartay hain, Python uska **bytecode .pyc file** mein save kar deta hai (usually __pycache__ folder mein).

🧪 Bytecode Example (with dis module)

```
python
CopyEdit
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age}
years old.")
```

```
person = Person("Arif Rozani", 20)
person.greet()
```

Agar hum bytecode dekhna chahein:

```
python
CopyEdit
import dis
dis.dis(Person)
```

Yeh code tumhein dikhayega ke Python ne har line ke liye kya bytecode generate kiya hai.

⚙️ Common Bytecode Instructions

Instruction	Matlab
LOAD_FAST	Variable uthana (e.g., name)
STORE_ATTR	Object ke andar attribute save karna (e.g., self.name = name)
LOAD_ATTR	Object se attribute lena
CALL	Function ya method call karna
RETURN_VALUE	Function se result wapas dena

❓ MCQs – Bytecode Practice

1. **What is Python Bytecode?**
   A) Machine Code
   B) High-level Source Code
   C) Intermediate Code

D) None of the above
✅ **Correct Answer:** C

2. **Where is bytecode stored after a Python file is executed?**
   A) .exe file
   B) .txt file
   C) .pyc file
   D) .log file
   ✅ **Correct Answer:** C

3. **Which module in Python is used to disassemble and view bytecode?**
   A) compile
   B) bytecode
   C) dis
   D) show
   ✅ **Correct Answer:** C

4. **What does the STORE_ATTR bytecode instruction do?**
   A) Stores a function
   B) Stores a variable globally
   C) Stores a value in an object's attribute
   D) Stores bytecode in memory
   ✅ **Correct Answer:**

🧾 What is a .pyc File?

.pyc file hoti hai **Python Compiled File** – is mein **bytecode** hota hai.

Jab bhi tum koi Python file **import** karte ho, Python us code ko pehle compile karta hai aur ek .pyc file bana deta hai jisme bytecode hota hai. Ye file automatically __pycache__ folder ke andar save hoti hai.

🛠️ How to Create a .pyc File

Aasan steps:

1. Ek Python file banao: hello.py

python

CopyEdit

```
def say_hello():
    print("Hello from .pyc world!")
```

2. Ab terminal (ya CMD) open karo aur yeh command likho:

bash

CopyEdit

```
python -m py_compile hello.py
```

3. Ab check karo:
    - Ek folder __pycache__ ban gaya hoga
    - Uske andar ek file hogi: hello.cpython-3X.pyc

🔍 What is dis.dis()?

dis ka matlab hai **Disassembler** – yeh Python ka ek built-in module hai jo Python ke code ka **bytecode** show karta hai.

🔎 How to Use dis.dis()

Example:

python

CopyEdit

```
def greet(name):
    print("Hello", name)


import dis
dis.dis(greet)
```

Output:

Yeh output kuch aisa dikhayega:

scss

CopyEdit

```
 2           0 LOAD_GLOBAL              0 (print)
             2 LOAD_CONST               1 ('Hello')
             4 LOAD_FAST                0 (name)
             6 CALL_FUNCTION            2
             8 RETURN_VALUE
```

Yeh line-by-line batata hai ke Python ne kya instructions generate ki hain.

✅ Summary – Step-by-Step

Task	Command / Code
.pyc file banana	python -m py_compile filename.py
Bytecode dekhna	import dis + dis.dis(function_name)

📦 Bonus Tip: .pyc File Ko Run Kaise Karein?

.pyc file ko direct run nahi karte usually, lekin agar karna chahein:

bash

CopyEdit

```
python __pycache__/hello.cpython-311.pyc
```

Lekin yaad rahe: is file ko run karne ka matlab hai ke tumhara code already compiled hai.

🧰 Introduction to the dis Module

dis Python ka ek **built-in module** hai jo kisi bhi function, method, class waghera ka **bytecode** nikaal kar dikhata hai. Iska use hum tab karte hain jab humein dekhna ho ke Python interpreter humare code ko **andar se kaise samajh kar chala raha hai**.

🤔 Why is Python Bytecode Important?

1. Platform Independence

Python ka bytecode har OS pe (Windows, Mac, Linux) chal sakta hai – **bus Python interpreter installed hona chahiye**.

2. Dynamic Typing

Python runtime pe decide karta hai ke variable ka type kya hai – compile time pe nahi.

3. Flexibility

Python ka bytecode modify bhi kiya ja sakta hai, aur interpreter ke behavior ko customize bhi kiya ja sakta hai (advance cheez hai ye, par possible hai).

⚙️ How Python Uses Bytecode

Step	Explanation
Compilation	Python script ko pehle compile kar ke bytecode banaya jata hai
Execution	Ye bytecode Python Virtual Machine (PVM) chalaati hai
Caching	Bytecode ko __pycache__ folder mein store kiya jata hai, taake agle run pe fast load ho

🔸 Indentation in Python

📘 What is Indentation?

Indentation ka matlab hai code ke aaghaz mein **space ya tab** dena – ye Python mein **block of code** dikhane ke liye use hota hai. Jaise function ke andar jo likha hai, ya if statement ke andar jo likhna hai – sab indent hona chahiye.

🤔 Why is Indentation Important?

Python mein { } brackets nahi hote block show karne ke liye – **sirf indentation hi batata hai ke kaunsa code kis block ka part hai**.

- Code ko **readable** banata hai
- **Structure clear** karta hai
- **Errors se bachata hai**

📏 Rules of Indentation:

✅ Use only **spaces** or only **tabs**, mix mat karo
✅ Standard = **4 spaces per level**
✅ Colon (:) ke baad wali line hamesha **indent** honi chahiye
✅ Functions & classes ke andar ka code indent karo

✅ Correct vs ❌ Incorrect Example:

python

CopyEdit

```
# ✅ Correct

if True:
```

```
    print("Hello")
    print("World")

# ❌ Incorrect
if True:
print("Hello")
  print("World")
```

🛠️ Practice Exercise:

```
python
CopyEdit
# ✅ Write a program using proper indentation:
x = int(input("Enter a number: "))

if x > 10:
    print("Greater than 10")
    print("This block is indented properly")
else:
    print("10 or less")
```

🔶 Python: Dynamically-Typed Language with Type Hinting

📚 Dynamically-Typed

Python mein variable ka type run-time pe decide hota hai.

python

CopyEdit

```
x = 5        # integer
x = "hello"  # ab string ho gaya
```

Python khud samajhta hai ke ab x kya hai.

🗒️ Optional Type Hinting

Python 3.5+ mein aaya **type hinting** – jisme tum bata sakte ho ke variable ya function ka expected type kya hai.

python

CopyEdit

```
age: int = 25
name: str = "Arif"
```

Functions ke liye:

python

CopyEdit

```
def greet(name: str) -> str:
    return "Hello " + name
```

⚙️ Example: Type Confusion

python

CopyEdit

```
age: int = input("Enter age: ")
print(f"Age is {age}")
print("Type is", type(age))  # str hoga, int nahi
```

Kyun? Kyunki input() hamesha string return karta hai.

🔍 Best Practices for Type Hinting

✅ Code ko **readable** banata hai
✅ IDEs mein **auto-complete** aur suggestions better hote hain
✅ mypy jaise tools se **static checking** ho sakti hai
✅ Code ka **documentation** better hota hai

📋 MCQs for Practice (English)

Q1: What does indentation in Python define?
A. Loop counters
B. Function names

C. Code structure ✅
D. Variable types

Q2: How many spaces are standard for one level of indentation in Python?
A. 2
B. 4 ✅
C. 8
D. None

Q3: What does the following code output?

python

CopyEdit

```
x = 5

if x > 3:

print("Yes")
```

A. Yes
B. Error ✅
C. 5
D. No output

Q4: What type is returned by input() in Python?
A. int
B. float
C. str ✅
D. bool

Q5: Which of these is a correct type hint?
A. x = 5: int
B. int: x = 5
C. x: int = 5 ✅
D. x = int: 5

🔷 Function Type Hinting in Python

👉 **Syntax:**

python

CopyEdit

```
def greet(name: str) -> str:
    return "Hello, " + name
```

📘 **Urdu Explanation:**
Yahan humne function ko bataya hai ke name ek string hoga, aur return bhi string karega.

🔹 Practice 1:

python

CopyEdit

```
def add(x: int, y: int) -> int:
    return x + y

print(add(2, 3))  # Output: 5
```

🔷 Complex Type Hinting

python

CopyEdit

```
my_list: list[int] = [1, 2, 3]

my_dict: dict[str, int] = {"a": 1, "b": 2}

my_tuple: tuple[str, int] = ("Ali", 22)
```

📘 **Urdu:**

- List mein integers hain
- Dictionary mein string key aur int value hai
- Tuple mein string aur int

🔹 Practice 2:

python

CopyEdit

```
students: list[str] = ["Ali", "Sara", "Zain"]

print(students[0])  # Output: Ali
```

🔷 Best Practices for Type Hinting

✅ Hamesha consistent raho
✅ Har function mein type hints do
✅ Complex types ka sahi use karo (list, dict, tuple)
✅ Any ka kam se kam use karo

🔷 4 Pillars of OOP in Python

Python **Object-Oriented Language** hai. Iska matlab hai ke ye 4 OOP pillars ko support karta hai:

Pillar Name	Urdu Mein Kya Matlab?
Encapsulation	Data aur methods ko class ke andar chhupa lena
Inheritance	Ek class dusri class ki properties use kare
Polymorphism	Ek hi method ka alag alag behavior
Abstraction	Sirf zaroori cheeze dikhana, detail chhupana

🔶 1. Encapsulation

python

CopyEdit

```
class Student:
    def __init__(self, name: str):
        self.name = name

    def show(self):
        print("Student:", self.name)
```

📘 self.name ko hum class ke andar hi access karte hain = **Encapsulation**

🔶 2. Inheritance

python

CopyEdit

```
class Animal:
    def speak(self):
        return "Animal sound"

class Dog(Animal):
    def speak(self):
        return "Bark"
```

📘 Dog class ne Animal class ke method ko use kiya = **Inheritance**

🔶 3. Polymorphism

python

CopyEdit

```
def make_sound(animal):
    print(animal.speak())


dog = Dog()
make_sound(dog)  # Output: Bark
```

📘 make_sound() function alag alag objects pe alag behave karega = **Polymorphism**

🔶 4. Abstraction

python

CopyEdit

```
from abc import ABC, abstractmethod


class Vehicle(ABC):
    @abstractmethod
```

```
    def start(self):
        pass

class Car(Vehicle):
    def start(self):
        print("Car started")
```

📘 Vehicle class ne sirf concept diya (abstract method), real kaam Car ne kiya = **Abstraction**

🔹 Practice 3 (Try yourself):

python

CopyEdit

```
# Create a class called Shape with abstract method area
# Inherit it in a class called Circle and implement area

# Hint: Use math.pi * r * r
```

✅ MCQs (English)

Q1: What does `name: str` mean in a function parameter?
A. It takes integer input
B. It takes string input ✅

C. It returns a string
D. It's a method

Q2: Which is a correct way to type hint a list of numbers?
A. `list[str]`
B. `list[int]` ✅
C. `dict[int]`
D. `tuple[str]`

Q3: Which is NOT a pillar of OOP?
A. Encapsulation
B. Composition ✅
C. Inheritance
D. Abstraction

Q4: What is `from abc import ABC, abstractmethod` used for?
A. Encapsulation
B. Creating objects
C. Abstraction ✅
D. Inheritance

Duck Typing in Python

Duck Typing Python ka aik bohot important concept hai. Ye programming approach Python ki flexibility ko bohot enhance karta hai, jisme aap kisi object ko uske type se zyada uske attributes aur methods ke hisaab se evaluate karte hain. Iska matlab ye hai ke agar aik object kisi specific behavior ko follow karta hai (jaise methods ya properties), toh aap usse us type ki tarah treat kar sakte hain, chahe wo us type ka na ho.

Duck Typing ka Mashhoor Quote:
"**Agar yeh ek duck ki tarah chalti hai aur baat karti hai, toh yeh ek duck hai.**"
Matlab agar kisi object mein required methods hain, toh us object ka asli type check karne ki zarurat nahi hoti.

Duck Typing ka Python mein Kaise Use Hota Hai?

Python mein duck typing kaafi common hai. Jab aap kisi object ke method ko call karte hain, toh Python yeh check karta hai ke us object mein woh method available hai ya nahi. Agar hai, toh operation perform ho jata hai, warna error aata hai.

Agar hum baat karein duck typing ka, toh Python yeh dekhne ki koshish karta hai ke object ne required behavior diya hai ya nahi. Agar object woh behavior provide karta hai, toh Python us object ko us type ka samajhta hai, chahe wo object asal mein us type ka ho ya nahi.

Example: Conversation Function (Duck Typing)

Maan lijiye, aap ek `have_conversation` function banate hain jo kisi bhi object ke saath baat kar sakta hai, bas us object ko `speak` method provide karna hoga.

python

CopyEdit

```
# Human Class

class Human:

    def speak(self):

        print("Human: I'm good, thanks!")


# Parrot Class
```

```
class Parrot:

    def speak(self):

        print("Parrot: Polly wants a cracker!")


# Conversation function

def have_conversation(person):

    print("\nhave_conversation: Hello, how are you? ", type(person))

    person.speak()


# Human aur Parrot ke objects

human = Human()

parrot = Parrot()


# Function call karna dono objects ke saath

have_conversation(human)   # Human se baat karna

have_conversation(parrot)  # Parrot se baat karna
```


Output:

vbnet

CopyEdit

```
have_conversation: Hello, how are you? <class '__main__.Human'>

Human: I'm good, thanks!
```

```
have_conversation: Hello, how are you? <class '__main__.Parrot'>

Parrot: Polly wants a cracker!
```

Explanation:

- have_conversation function ko kis type ka object pass kiya gaya hai, uska koi farq nahi padta.
- Yeh function sirf is baat ko dekh raha hai ke jo object pass kiya gaya hai, usmein speak method hai ya nahi. Agar hai, toh usse call kar lega.
- Is example mein, hum Human aur Parrot classes ke objects pass kar rahe hain.
- Dono classes mein speak method hai, toh Python dono objects se baat karne mein kamyab hota hai.

Duck Typing ki Tareeqe Se New Types ko Add Karna

Aap koi naya object create kar sakte hain jo speak method ko implement kare, aur yeh function us object ke saath bhi kaam karega bina kisi problem ke.

Example: Robot Class ka Example

python

CopyEdit

```
# Robot Class

class Robot:

    def speak(self):

        print("Robot: Beep boop, I am functioning within normal parameters!")
```

```
# Robot ka object banate hain

robot = Robot()


# Conversation function ke saath Robot ko pass karna

have_conversation(robot)  # Output: Robot: Beep boop, I am functioning within normal parameters!
```

Output:

```
vbnet

CopyEdit

have_conversation: Hello, how are you? <class '__main__.Robot'>

Robot: Beep boop, I am functioning within normal parameters!
```

Key Points:

- **Duck Typing** ka concept Python mein aise kaam karta hai ke agar koi object required behavior (methods/attributes) provide karta hai, toh aap usse us type ki tarah treat kar sakte hain.
- Is approach se nayi classes ko bina kisi modification ke add kiya jaa sakta hai.
- Python ka yeh feature code ko zyada flexible aur extendable banata hai.

Duck Typing ki Misaal:

Jab aap Python mein kisi method ko call karte hain, toh Python **type check nahi karta**, bas yeh dekhta hai ke object mein wo method hai ya nahi. Agar hai, toh wo method call ho jati hai, agar nahi hai toh error raise hota hai. Is tarah se Python bohot flexible hota hai aur aapko type ko explicitly define karne ki zarurat nahi hoti.

Duck Typing ka Concept Zyada Achhe Se Samajhne Ki Example:

Jese humne pichli baar "Duck Typing" ke baare mein kaha tha, uska basic idea yeh hai:

"Agar yeh ek duck ki tarah chalti hai aur baat karti hai, toh yeh ek duck hai."

Matlab agar ek object kisi required behavior ko perform karta hai, toh uska type check karne ki zarurat nahi hai. Hum sirf is baat ko dekhte hain ke kya wo object us behavior ko implement karta hai ya nahi.

Example: Duck Typing in Action

Maan lijiye humare paas `Car` aur `Boat` objects hain, dono ko hum ek function ke through evaluate karte hain. Yeh function har object ke `drive()` method ko call karega. Agar kisi object ke paas `drive()` method hai, toh wo object function ke saath kaam karega.

Code Example:

python

CopyEdit

```
class Car:

    def drive(self):

        print("Car is driving on the road!")
```

```
class Boat:
    def drive(self):
        print("Boat is sailing on the water!")

class Animal:
    def speak(self):
        print("Animal is making noise!")

# Function to interact with objects that have 'drive' method
def start_journey(vehicle):
    vehicle.drive()  # Function only cares if the object can 'drive'

# Objects creation
car = Car()
boat = Boat()
animal = Animal()

# Calling the function with different objects
start_journey(car)   # Output: Car is driving on the road!
start_journey(boat)  # Output: Boat is sailing on the water!
# start_journey(animal)  # This will give an error as 'drive' method
is not present in Animal class
```

Output:

csharp

CopyEdit

```
Car is driving on the road!

Boat is sailing on the water!
```

Explanation:

1. **Car** aur **Boat** dono classes mein `drive()` method hai, toh hum unko `start_journey()` function mein pass kar sakte hain.

2. Jab hum `start_journey(car)` call karte hain, toh `Car` object ke `drive()` method ko call kar liya jata hai. Is case mein, function ko yeh farq nahi padta ke object ka type kya hai, bas wo `drive()` method ke existence ko check karta hai.

3. Agar hum `start_journey(animal)` call karte, toh error aata, kyunki `Animal` class mein `drive()` method nahi hai. Yeh humein batata hai ke Duck Typing mein agar required behavior (yani `drive()` method) object mein nahi hai, toh Python error de dega.

Duck Typing ka Tareeqa:

- Jab hum `start_journey()` function ko call karte hain, Python yeh check karta hai ke object mein `drive()` method hai ya nahi.

- Agar method hai, toh us object ko function ka part samajh ke uska `drive()` method call kar leta hai.

- Agar method nahi hai, toh error raise hota hai, lekin Python ko object ka type check karne ki zarurat nahi padti.

Duck Typing ka Ek Aur Example:

Hum koi bhi object bana sakte hain, jab tak uske paas `speak()` method hai, wo object function ke saath kaam karega.

Code Example:

python

CopyEdit

```
class Human:
    def speak(self):
        print("Human: Hello, how are you?")


class Parrot:
    def speak(self):
        print("Parrot: Squawk! Polly wants a cracker!")


class Robot:
    def speak(self):
        print("Robot: Beep boop, I am functioning!")


def have_conversation(speaker):
    speaker.speak()


# Objects
human = Human()
```

```
parrot = Parrot()

robot = Robot()


# Function calls

have_conversation(human)   # Output: Human: Hello, how are you?

have_conversation(parrot)  # Output: Parrot: Squawk! Polly wants a
cracker!

have_conversation(robot)   # Output: Robot: Beep boop, I am
functioning!
```

Output:

```
makefile

CopyEdit

Human: Hello, how are you?

Parrot: Squawk! Polly wants a cracker!

Robot: Beep boop, I am functioning!
```

Explanation:

- **Human**, **Parrot**, aur **Robot** sabhi classes mein `speak()` method hai.
- Jab hum **have_conversation** function ko call karte hain, Python ko yeh farq nahi padta ke object ka type kya hai. Function sirf yeh dekh raha hai ke `speak()` method available hai ya nahi.
- Yeh function sabhi objects ke saath kaam karta hai, jo `speak()` method ko implement karte hain.

Key Takeaways:

- **Duck Typing** ka concept aapko objects ke type se zyada unke behavior ko dekhne ka mauka deta hai.
- Agar kisi object mein required behavior (methods/attributes) hai, toh aap us object ko us type ka samajh sakte hain, chahe wo asli mein us type ka ho ya nahi.
- Python mein aise flexibility ki wajah se code zyada reusable aur flexible ho jata hai.

1. Indentation in Python

1. **What is the primary purpose of indentation in Python?**
    - a) To define loops
    - b) To define the structure of the code
    - c) To create functions
    - d) To create variables
2. **Answer:** b) To define the structure of the code
3. **What happens if you do not use proper indentation in Python?**
    - a) The program will still run but it will be messy
    - b) SyntaxError will be raised
    - c) The code will not compile
    - d) No impact

4. **Answer:** b) SyntaxError will be raised

5. **What is the correct way to indent in Python?**
    - a) 4 spaces
    - b) 2 spaces
    - c) 1 tab
    - d) 3 spaces

6. **Answer:** a) 4 spaces

7. **What will happen if you mix tabs and spaces for indentation?**
    - a) It will work fine
    - b) IndentationError will occur
    - c) The code will throw a runtime error
    - d) The program will execute correctly

8. **Answer:** b) IndentationError will occur

---

2. Python's Dynamically-Typed Language with Optional Type Hinting

5. **Which of the following is true about Python?**
    - a) Python is statically typed
    - b) Python supports optional type hinting

- c) Python variables must have defined data types
- d) Python is compiled language

6. **Answer:** b) Python supports optional type hinting

7. **What is type hinting in Python?**
   - a) A way to check data types at runtime
   - b) A way to define expected data types for variables and function parameters
   - c) A Python error message
   - d) None of the above

8. **Answer:** b) A way to define expected data types for variables and function parameters

What is the output of the following code?

```
 python
CopyEdit
age: int = "22"

print(type(age))
```

9.
   - a) `<class 'int'>`
   - b) `<class 'str'>`
   - c) `<class 'float'>`
   - d) `Error`

10. **Answer:** b) `<class 'str'>`

11. **Which of the following syntax is correct for type hinting a function parameter?**

    - a) `def greet(name: string) -> str:`
    - b) `def greet(name: str) -> string:`
    - c) `def greet(name: str):`
    - d) `def greet(name: str) -> str:`

12. **Answer:** d) `def greet(name: str) -> str:`

13. **What is the default data type of a variable in Python?**

    - a) Static
    - b) String
    - c) Dynamic
    - d) Integer

14. **Answer:** c) Dynamic

---

3. Function with Type Hints

10. **What is the correct way to specify that a function returns an integer in Python type hints?**

    - a) `-> int`
    - b) `-> integer`

- c) `int ->`
- d) `return int`

11. **Answer:** a) `-> int`

12. **Which of the following is an example of type hinting for a list of strings in Python?**

- a) `List[int]`
- b) `List[str]`
- c) `str[List]`
- d) `List[str()]`

13. **Answer:** b) `List[str]`

What is the return type of the following function?

```
 python
CopyEdit
def sum_numbers(a: int, b: int) -> int:
    return a + b
```

14.

- a) int
- b) None
- c) str
- d) float

15. **Answer:** a) int

16. **What is the purpose of type hinting in Python?**
    - a) To help Python execute code faster
    - b) To check data types at compile time
    - c) To make code easier to understand
    - d) To prevent type errors at runtime
17. **Answer:** c) To make code easier to understand

---

4. Object-Oriented Programming (OOP) Methods

14. **Which of the following is NOT an OOP principle in Python?**
    - a) Encapsulation
    - b) Inheritance
    - c) Polymorphism
    - d) Static typing
15. **Answer:** d) Static typing
16. **What does the `super()` function do in Python?**
    - a) It calls a method from the parent class
    - b) It calls a method from a sibling class
    - c) It defines a method in the parent class
    - d) It does nothing

17. **Answer:** a) It calls a method from the parent class

18. **Which of the following demonstrates polymorphism?**

    - a) A class method is overridden in the subclass
    - b) An instance variable is shared by all instances
    - c) A method is defined only in the parent class
    - d) A method is hidden by the subclass

19. **Answer:** a) A class method is overridden in the subclass

20. **What is the term for data hiding in object-oriented programming?**

    - a) Encapsulation
    - b) Abstraction
    - c) Polymorphism
    - d) Inheritance

21. **Answer:** a) Encapsulation

22. **Which class would you use to define a parent class?**

    - a) class Parent:
    - b) class Super:
    - c) class Child:
    - d) class Derived:

23. **Answer:** a) class Parent:

---

5. Duck Typing

19. **What is the main idea of duck typing in Python?**

    - a) If an object has the methods you need, you can use it, regardless of its class.
    - b) Objects must be explicitly typed before use.
    - c) Objects are checked for type before use.
    - d) Objects can only be used if they are of the correct class.

20. **Answer:** a) If an object has the methods you need, you can use it, regardless of its class.

21. **What will happen if you try to use a method that an object does not have in Python?**

    - a) An exception is raised
    - b) Python will ignore it and continue execution
    - c) The program will crash
    - d) It will execute but return `None`

22. **Answer:** a) An exception is raised

23. **Which of the following is an example of duck typing?**

    - a) Using `hasattr()` to check if an object has a method

- b) Checking the type of an object with `type()`
- c) Using `isinstance()` to check if an object is of a particular class
- d) Calling methods on an object without checking its type

24. **Answer:** d) Calling methods on an object without checking its type

25. **Which of the following animals can be used as an example for duck typing?**

- a) A fish that swims like a duck
- b) A dog that barks like a cat
- c) A parrot that talks like a human
- d) A cat that flies like a bird

26. **Answer:** c) A parrot that talks like a human

---

6. First-Class Functions

23. **Which statement is true about functions in Python?**

- a) Functions cannot be assigned to variables
- b) Functions can be passed as arguments to other functions
- c) Functions cannot be returned from other functions
- d) Functions are not considered first-class objects

24. **Answer:** b) Functions can be passed as arguments to other functions

25. **In Python, what is the behavior of a function that is assigned to a variable?**

    - a) It stops functioning
    - b) It can be called using that variable
    - c) It becomes a string
    - d) It becomes a list

26. **Answer:** b) It can be called using that variable

27. **Which is the correct way to pass a function as an argument in Python?**

    - a) `call_function(function())`
    - b) `call_function(function)`
    - c) `function(call_function())`
    - d) `call_function(function())`

28. **Answer:** b) `call_function(function)`

29. **Which of the following Python functions can be passed as an argument?**

    - a) `def add(x, y): return x + y`
    - b) `def add(x, y): return x - y`
    - c) `add`

- d) All of the above

30. **Answer:** d) All of the above

---

7. Type Hinting for Complex Types

27. **Which of the following is the correct type hint for a list of integers?**

    - a) `list[int]`
    - b) `list[str]`
    - c) `List[int]`
    - d) `int[List]`

28. **Answer:** c) `List[int]`