**1. Erklären Sie Instanz, Referenz, Referenzzähler, Garbage Collection und del.**

* **Instanz**: Ein konkretes Objekt, das von einer Klasse erzeugt wird. Beispiel: obj = Klasse().
* **Referenz**: Ein Verweis auf ein Objekt im Speicher. Mehrere Referenzen können auf dasselbe Objekt zeigen.
* **Referenzzähler**: Python verwendet einen Referenzzähler, um festzustellen, wie viele Referenzen auf ein Objekt zeigen. Sinkt der Zähler auf null, wird das Objekt gelöscht.
* **Garbage Collection**: Automatischer Prozess, der ungenutzte Objekte entfernt, um Speicher freizugeben.
* **del**: Löscht eine Referenz zu einem Objekt. Das Objekt wird jedoch nur dann aus dem Speicher entfernt, wenn keine weiteren Referenzen existieren.

**2. Was sind Type Hints?**

Type Hints erlauben es, den erwarteten Datentyp von Variablen und Funktionsparametern zu definieren. Beispiel:

def add\_numbers(a: int, b: int) -> int:

return a + b

**3. Erklären Sie den Unterschied zwischen Compiler- und Interpretersprachen.**

* **Compiler-Sprachen** (z. B. C++): Übersetzen den gesamten Code vor der Ausführung in Maschinencode.
* **Interpreter-Sprachen** (z. B. Python): Übersetzen und führen Code zur Laufzeit Zeile für Zeile aus.

**4. Unterschiede zwischen Java und Assembler.**

* **Java**: Plattformunabhängig, objektorientiert, höherer Abstraktionsgrad, Garbage Collection.
* **Assembler**: Plattformabhängig, arbeitet auf niedriger Ebene, direkte Kontrolle über Hardware, keine automatische Speicherverwaltung.

**5. Unterschied if-elif-else und match case.**

* **if-elif-else**: Kontrollstruktur, die Bedingungen prüft.
* **match case**: Musterabgleich (ab Python 3.10). Eignet sich besser für komplexe Vergleiche:
* match value:
* case 1:
* print("Eins")
* case \_:
* print("Andere Zahl")

**6. Was sind Kontrollstrukturen? Welche eignet sich, um viele Möglichkeiten eines Inhalts zu überprüfen?**

* **Kontrollstrukturen**: Mechanismen wie Schleifen und Verzweigungen (if, for, while, match).
* **Empfehlung**: match case für viele Bedingungen oder ein Wörterbuch als Lookup.

**7. Unterschied zwischen iterativer und rekursiver Programmierung.**

* **Iterativ**: Probleme mit Schleifen lösen.
* **Rekursiv**: Eine Funktion ruft sich selbst auf.
* **Beispiel**: Faktorial iterativ vs. rekursiv:
* def factorial\_iter(n):
* result = 1
* for i in range(1, n + 1):
* result \*= i
* return result
* def factorial\_rec(n):
* return 1 if n == 0 else n \* factorial\_rec(n - 1)

**8. Nennen Sie drei fachliche Unterschiede zwischen Java und Python.**

1. Python benötigt keine Typdeklarationen (dynamisch vs. statisch typisiert).
2. Python unterstützt multiple Vererbung direkt, Java nicht.
3. Python ist interpreterbasiert, Java verwendet JVM.

**9. Erklären Sie mutable und immutable, sowie deren Vor- und Nachteile.**

* **Mutable**: Veränderbare Objekte (z. B. Listen). Vorteil: Effizient bei Änderungen. Nachteil: Nebenwirkungen möglich.
* **Immutable**: Unveränderbare Objekte (z. B. Tupel). Vorteil: Sichere, unveränderbare Daten. Nachteil: Änderungen erfordern Kopien.

**10. Erklären Sie \_\_name\_\_ in Python.**

Das \_\_name\_\_-Attribut gibt an, ob ein Skript direkt ausgeführt wird (\_\_main\_\_) oder importiert wurde (Modulname).

**11. Unterschied zwischen Arrays in Python und Java.**

* **Python**: Arrays können mit der Bibliothek array erstellt werden, Listen sind flexibler. Typenprüfung ist schwächer.
* **Java**: Arrays sind fester Bestandteil der Sprache, erfordern Typenangaben, feste Größe.

**12. Prozesse, Threads allgemein in Betriebssystemen.**

* **Prozesse**: Unabhängige Programme mit eigenem Speicherbereich.
* **Threads**: Leichte Prozesse, teilen Speicher mit anderen Threads desselben Prozesses.

**13. Unterschied zwischen array und list in Python sowie die Vorteile von Arrays.**

* **Arrays**: Weniger flexibel, für numerische Daten optimiert.
* **Listen**: Beliebige Datentypen, dynamische Größe.
* **Vorteil von Arrays**: Speicherplatz- und Rechengeschwindigkeitsoptimierungen.

**14. Nennen Sie drei Vorteile von Python gegenüber C-ähnlichen Sprachen.**

1. Dynamische Typisierung.
2. Kürzere Syntax, leichter lesbar.
3. Umfassende Standardbibliothek.

**15. Beispiele für mutable und immutable Datentypen in Python, und wann welche sinnvoll sind.**

* **Mutable**: Liste, Wörterbuch. Verwendung bei häufigen Änderungen.
* **Immutable**: Tupel, Zeichenketten. Verwendung für konstant bleibende Daten.

**16. Drei Gemeinsamkeiten zwischen Java und Python.**

1. Objektorientierung.
2. Plattformunabhängigkeit (Python: Interpreter, Java: JVM).
3. Unterstützung für umfangreiche Bibliotheken.

**17. Erklären Sie die pass-Anweisung in Python.**

pass ist ein Platzhalter für leere Codeblöcke. Beispiel:

if condition:

pass # Platzhalter für späteren Code

**18. Unterschied im Umgang mit Datentypen zwischen Java und Python, einschließlich Type Hints.**

* **Java**: Strenge Typisierung, Datentypen zur Compile-Zeit geprüft.
* **Python**: Dynamisch typisiert, optionale Type Hints.

**19. List Comprehension Beispiele.**

* Quadrate positiver Zahlen:
* [x\*\*2 for x in range(1, 11)]
* Positive gerade Zahlen:
* [x for x in range(1, 11) if x % 2 == 0]
* Liste mit jedem zweiten Element (nicht None):
* [x for x in lst[::2] if x is not None]

**20. Floor-Division in Python 3.**

Verwendet // für ganzzahlige Division. Beispiel:

7 // 3 # Ergebnis: 2

**21. Garbage Collection allgemein.**

Der automatische Prozess zur Speicherbereinigung durch Entfernen nicht erreichbarer Objekte.

**22. Unterschied zwischen tuple, range und set.**

* **Tuple**: Unveränderbare Sammlung.
* **Range**: Sequenz von Zahlen, speicherfreundlich.
* **Set**: Ungeordnete Sammlung einzigartiger Elemente.

**23. Unterschied zwischen copy und deepcopy.**

* **copy**: Erstellt eine flache Kopie.
* **deepcopy**: Kopiert auch verschachtelte Objekte vollständig.

**24. Was sind Dunder-Variablen und -Methoden?**

* **Dunder-Variablen/-Methoden**: Beginnen und enden mit \_\_ (z. B. \_\_init\_\_, \_\_name\_\_). Werden intern von Python verwendet.

**25. f-strings nach Vorgabe erstellen.**

name = "Alice"

age = 30

print(f"Name: {name}, Alter: {age}")

value = 12345.6789

formatted = f"{value:,.2f}"

print(formatted) # Ausgabe: 12,345.68

**26. Was ist bei unittest mit „automatische“ Tests gemeint?**

Tests werden automatisiert ausgeführt und prüfen erwartete Ergebnisse gegen tatsächliche Resultate.

**27. Warum ist es ratsam, unittest vor der Codeentwicklung zu schreiben?**

Tests definieren Anforderungen und erleichtern die Fehlersuche (Test-Driven Development).

**28. Beispiel für den ternären Operator.**

Konventionell:

if x > 0:

result = "positiv"

else:

result = "negativ"

Ternär:

result = "positiv" if x > 0 else "negativ"

**29. Was ist pdb und wann sinnvoll?**

Python-Debugger, nützlich für interaktive Fehleranalyse.

**30. Exceptions anhand eines vollständigen Beispiels:**

try:

x = int(input("Geben Sie eine Zahl ein: "))

except ValueError:

print("Ungültige Eingabe.")

else:

print(f"Sie haben {x} eingegeben.")

finally:

print("Fertig.")

**31. Unterschied def und lambda.**

* **def**: Erstellt eine benannte Funktion mit mehreren Zeilen.
* **lambda**: Erstellt anonyme Einzeilenfunktionen. Beispiel:

# Mit def

def square(x):

return x\*\*2

# Mit lambda

square = lambda x: x\*\*2

**32. Sinn und Vorteile von Virtual Environments in Python.**

* **Sinn**: Isolation von Projektabhängigkeiten.
* **Vorteile**: Keine Konflikte zwischen Bibliotheken verschiedener Projekte.

**33. Nutzen von map und lambda, um Listen zu verarbeiten.**

numbers = [1, 2, 3]

squared = map(lambda x: x\*\*2, numbers) # Ergebnis: [1, 4, 9]

**34. Zwei Methoden zur Erstellung der requirements.txt in Virtual Envs.**

1. Manuelles Schreiben der Abhängigkeiten.
2. Automatisch:

pip freeze > requirements.txt

**35. Beispiel einer Funktion mit nur Schlüsselwortpaaren als Parameter.**

def greet(\*, name: str):

print(f"Hallo, {name}!")

greet(name="Alice")

**36. Bedeutung von self in Methoden bei Vererbungsstrukturen.**

self verweist auf die Instanz der Klasse, die die Methode aufruft.

**37. Erklären Sie super() und Unterschiede zu Java.**

* **Python**: Greift auf die Methode der Elternklasse zu.
* **Besonderheit**: Python benötigt keinen Konstruktor-Aufruf wie in Java.

class Parent:

def greet(self):

print("Hallo von der Elternklasse!")

class Child(Parent):

def greet(self):

super().greet() # Ruft die Methode der Elternklasse auf

print("Hallo von der Kindklasse!")

child\_instance = Child()

child\_instance.greet()

**38. Unterschied zwischen Klasse und Objekt.**

* **Klasse**: Bauplan.
* **Objekt**: Instanz der Klasse.

**39. Was ist \_\_init\_\_ in Python?**

Spezialmethode, um die Initialisierung einer Instanz zu definieren.

**40. Unterschied zwischen Instanzvariable und Klassenvariable.**

* **Instanzvariable**: Gehört zu einer Instanz, z. B. self.name.
* **Klassenvariable**: Gehört zur Klasse, z. B. ClassName.variable.

**41. Sichtbarkeitsmodifikatoren in Python im Vergleich zu Java.**

* Python hat keine strengen Modifikatoren.
* \_protected: Konvention für geschützte Attribute.
* \_\_private: Name Mangling für privaten Zugriff.

**42. Wie „versteckt“ man etwas in Python?**

Durch Name Mangling oder Konventionen (\_\_name für privat).

**43. Unterschied \_\_str\_\_ und \_\_repr\_\_.**

* **\_\_str\_\_**: Nutzerfreundliche Darstellung.
* **\_\_repr\_\_**: Entwicklerfreundliche Darstellung.

**44. Was passiert, wenn ein Attribut auf ein Objekt geschrieben wird, das nicht in der Klasse definiert ist?**

Ein neues Attribut wird dynamisch zur Instanz hinzugefügt.

**45. @staticmethod und @classmethod Unterschiede.**

* **@staticmethod**: Funktion, die weder Instanz- noch Klassenbezug hat.
* **@classmethod**: Funktion mit Zugriff auf Klassenattribute (cls).

**46. Was ist Polymorphismus?**

Die Fähigkeit, verschiedene Klassen über dieselbe Schnittstelle zu nutzen.

**47. Was ist ein MRO (Method Resolution Order)?**

Definiert die Reihenfolge, in der Klassenmethoden aufgerufen werden. Kann mit ClassName.mro() geprüft werden.

**48. Was sind Lambdafunktionen?**

Anonyme Einzeilenfunktionen, oft in Kombination mit map, filter und reduce verwendet.

print((lambda x, y: x \* y)(4, 5)) # Ausgabe: 20

**49. Zwei Unterschiede in der objektorientierten Programmierung zwischen Java und Python.**

1. Python unterstützt Mehrfachvererbung direkt.
2. Python benötigt keine Typdeklaration für Variablen.

**50. Python-Namespaces, Erklärung zu global und local.**

* **Namespace**: Sammlung von Namen und Objekten.
* **Global**: Gültig im gesamten Modul.
* **Local**: Gültig innerhalb einer Funktion.