Zod Practice Questions & Answers

1. Define a basic string schema and parse a valid value.

```
const schema = z.string();
schema.parse("hello");
// Output: "hello"
```

2. Validate a number is positive.

```
const schema = z.number().positive();
schema.parse(-5);
// Throws error: not a positive number
```

3. Use .safeParse() instead of .parse().

```
const schema = z.number();
schema.safeParse("123");
// Output: { success: false, error: [ZodError] }
```

4. Create an object schema with name (string) and age (number).

```
const schema = z.object({ name: z.string(), age: z.number() });
schema.parse({ name: "Alice", age: 30 });
```

5. Make an optional field.

```
const schema = z.object({ email: z.string().email().optional() });
schema.parse({}); // Valid
```

6. Use .nullable()

```
const schema = z.string().nullable();
schema.parse(null); // Valid
```

7. Array of numbers greater than 10.

```
const schema = z.array(z.number().gt(10));
schema.parse([11, 20, 30]); // Valid
```

8. Enums using z.enum()

```
const schema = z.enum(["admin", "user", "guest"]);
schema.parse("superuser"); // Throws error
```

9. Use .refine() for custom validation.

```
const schema = z.string().refine(val => val.length >= 8, { message: "Too short" });
```

10. Use .transform() to convert input

```
const schema = z.string().transform(val => val.toUpperCase());
schema.parse("hello"); // "HELLO"
```

11. Use .default()

```
const schema = z.string().default("guest");
schema.parse(undefined); // "guest"
```

12. Coerce a string to a number

```
const schema = z.coerce.number();
schema.parse("42"); // 42
```

13. Use .nonempty() on strings

```
const schema = z.string().nonempty();
schema.parse(""); // Error
```

14. Nested object schema

```
const schema = z.object({ user: z.object({ id: z.number(), name: z.string() }) });
```

15. Schema merging

```
const a = z.object({ name: z.string() });
const b = z.object({ age: z.number() });
a.merge(b).parse({ name: "A", age: 20 });
```

16. Discriminated union

```
const schema = z.discriminatedUnion("type", [
  z.object({ type: z.literal("circle"), radius: z.number() }),
  z.object({ type: z.literal("square"), side: z.number() })
]);
```

17. Catch all additional fields

```
const schema = z.object({ name: z.string() }).passthrough();
schema.parse({ name: "A", age: 10 }); // Valid
```

18. .strict() to disallow unknown fields

```
const schema = z.object({ name: z.string() }).strict();
schema.parse({ name: "A", age: 10 }); // Error
```

19. .pick() and .omit()

```
const schema = z.object({ a: z.string(), b: z.number(), c: z.boolean() });
schema.pick({ a: true, c: true });
```

20. Infer TypeScript types using .infer

```
const schema = z.object({ name: z.string(), age: z.number() });
type User = z.infer<typeof schema>;
```

21. z.literal() for exact values

```
const schema = z.literal("open");
schema.parse("open"); // Valid
schema.parse("closed"); // Error
```

22. z.union() to allow multiple schemas

```
const schema = z.union([z.string(), z.number()]);
schema.parse("text"); // Valid
schema.parse(42); // Valid
```

23. z.tuple() for fixed-length arrays

```
const schema = z.tuple([z.string(), z.number()]);
schema.parse(["hello", 5]); // Valid
```

24. z.record() for object with dynamic keys

```
const schema = z.record(z.string());
schema.parse({ foo: "bar" }); // Valid
```

25. z.set() to validate JavaScript Sets

```
const schema = z.set(z.number());
schema.parse(new Set([1, 2, 3])); // Valid
```