1. TypeScript Basics

Q1. What will be the output of the following code snippet?

```typescript
let isDone: boolean = false;
console.log(isDone);
```

A. `true`

B. `false`

C. `undefined`

D. `null`

Answer: B. `false`

2. Type Annotations

Q2. What is the type of `x` in the following code?

```typescript
let x: number = 10;
console.log(typeof x);
```

A. `string`

B. `number`

C. `boolean`

D. `object`

Answer: B. `number`

3. Interfaces

Q3. What does the following code print?

```typescript
interface Person {
  name: string;
  age: number;
}

let person: Person = { name: "John", age: 30 };
console.log(person.name);
```

A. `John`

B. `30`

C. `undefined`

D. `null`

Answer: A. `John`

4. Classes
Q4. What is the output of this code?
```typescript
class Animal {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
}

let dog = new Animal("Dog");
console.log(dog.name);
```
A. `Dog`
B. `dog`
C. `undefined`
D. `null`

Answer: A. `Dog`

5. Access Modifiers

Q5. What will be the output of the following code?

```typescript
class Person {
  private name: string;
  constructor(name: string) {
    this.name = name;
  }
  getName(): string {
    return this.name;
  }
}

let person = new Person("Alice");
console.log(person.getName());
```

A. `Alice`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Alice`

6. Readonly Modifier

Q6. What is the error in this code?

```typescript
class Car {
  readonly model: string;
  constructor(model: string) {
    this.model = model;
  }
}

let car = new Car("Toyota");
car.model = "Honda";
```

A. `model is not defined`

B. `Cannot assign to 'model' because it is a read-only property`

C. `Constructor must initialize readonly properties`

D. `No error`

Answer: B. `Cannot assign to 'model' because it is a read-only property`

7. Inheritance

Q7. What will be the output of the following code?

```typescript
class Animal {
  move() {
    console.log("Moving...");
  }
}

class Dog extends Animal {
  bark() {
    console.log("Barking...");
```

  }
}

let dog = new Dog();
dog.bark();
dog.move();
~~~

A. `Moving... Barking...`

B. `Barking... Moving...`

C. `Moving...`

D. `Barking...`

Answer: B. `Barking... Moving...`

8. Generics

Q8. What does the following code print?

```typescript
function identity<T>(arg: T): T {
  return arg;
}
~~~

console.log(identity<number>(10));

A. `10`

B. `undefined`

C. `null`

D. `Error`

**Answer:** A. `10`

### 9. Enums
#### Q9. What is the output of the following code?
```typescript
enum Color {
  Red,
  Green,
  Blue
}

let c: Color = Color.Green;
console.log(c);
```

A. `0`

B. `1`

C. `2`

D. `Green`

Answer: B. `1`

10. Type Assertions

Q10. What does the following code print?

```typescript
let someValue: any = "Hello World";
let strLength: number = (someValue as string).length;
console.log(strLength);
```

A. `10`

B. `11`

C. `12`

D. `Error`

Answer: B. `11`

11. Tuples

Q11. What will be the output of the following code?

```typescript
let x: [string, number];
x = ["hello", 10];
console.log(x[0].substr(1));
```

A. `h`

B. `ello`

C. `lo`

D. `Error`

Answer: B. `ello`

12. Union Types

Q12. What is the output of the following code?

```typescript
function formatCommandLine(command: string[] | string) {
  let line = "";
  if (typeof command === "string") {
    line = command.trim();
  } else {
    line = command.join(" ").trim();
  }
  return line;
}

console.log(formatCommandLine("  Hello  "));
```
A. `Hello`

B. `  Hello  `

C. `Error`

D. `undefined`

Answer: A. `Hello`

13. Intersection Types

Q13. What does the following code print?

```typescript
```

```
interface ErrorHandling {
  success: boolean;
  error?: { message: string };
}

interface ArtworksData {
  artworks: { title: string }[];
}

type ArtworksResponse = ArtworksData &
ErrorHandling;

const handleResponse = (response:
ArtworksResponse) => {
  if (response.success) {
    console.log(response.artworks);
  } else {
    console.log(response.error.message);
  }
};
```

handleResponse({ success: true, artworks: [{ title: "Mona Lisa" }] });

A. `Mona Lisa`

B. `[{ title: "Mona Lisa" }]`

C. `undefined`

D. `Error`

Answer: B. `[{ title: "Mona Lisa" }]`

14. Literal Types

Q14. What will be the output of the following code?

```typescript
let x: "hello" = "hello";
console.log(x);
```

A. `hello`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `hello`

15. Functions
Q15. What is the output of this code snippet?
```typescript
function add(a: number, b: number): number {
  return a + b;
}

console.log(add(5, 3));
```
A. `8`

B. `5`

C. `3`

D. `Error`

Answer: A. `8`

16. Optional Parameters
Q16. What is the output of the following code?

```typescript
function buildName(firstName: string, lastName?: string) {
  return lastName ? `${firstName} ${lastName}` : firstName;
}

console.log(buildName("John"));
```

A. `John`

B. `John undefined`

C. `undefined`

D. `Error`

Answer: A. `John`

17. Default Parameters

Q17. What does the following code print?

```typescript
function buildName(firstName: string, lastName = "Doe") {
```

  return `${firstName} ${lastName}`;
}

console.log(buildName("John"));
~~~

A. `John Doe`

B. `John undefined`

C. `John`

D. `Error`

Answer: A. `John Doe`

18. Rest Parameters

Q18. What will be the output of the following code?

```typescript
function buildName(firstName: string, ...restOfName: string[]) {
  return `${firstName} ${restOfName.join(" ")}`;
}
~~~

console.log(buildName("John", "Doe", "Smith"));

A. `John Doe Smith`

B. `John`

C. `John Doe`

D. `Error`

**Answer:** A. `John Doe Smith`

### 19. Arrow Functions
#### Q19. What is the output of the following code?
```typescript
let add = (a: number, b: number): number => a + b;
console.log(add(5, 3));
```

A. `8`

B. `5`

C. `3`

D. `Error`

Answer: A. `8`

20. Destructuring

Q20. What will be the output of this code snippet?

```typescript
let input = [1, 2];
let [first, second] = input;
console.log(first, second);
```

A. `1 2`

B. `2 1`

C. `undefined`

D. `Error`

Answer: A. `1 2`

21. Spread Operator

Q21. What does the following code print?

```typescript
let arr1 = [1, 2];
let arr2 = [...arr1, 3, 4];
```

console.log(arr2);

A. `[1, 2, 3, 4]`

B. `[3, 4, 1, 2]`

C. `[1, 2]`

D. `Error`

Answer: A. `[

1, 2, 3, 4]`

22. Type Guards

Q22. What is the output of this code snippet?

```typescript
function isString(x: any): x is string {
  return typeof x === "string";
}

console.log(isString("Hello"));
```

A. `true`

B. `false`

C. `undefined`

D. `Error`

Answer: A. `true`

23. Async/Await
Q23. What does the following code print?
```typescript
async function fetchData() {
  return "Data fetched";
}

fetchData().then(console.log);
```
A. `Data fetched`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Data fetched`

24. Promises

Q24. What will be the output of the following code?

```typescript
let promise = new Promise((resolve, reject) => {
  resolve("Success");
});

promise.then((message) => {
  console.log(message);
});
```

A. `Success`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Success`

25. Modules

Q25. What does the following code print?
```typescript
// module.ts
export const name = "TypeScript";

// main.ts
import { name } from "./module";
console.log(name);
```
A. `TypeScript`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `TypeScript`

26. Namespaces
Q26. What is the output of this code snippet?
```typescript
namespace MyNamespace {
  export function sayHello() {
```

    return "Hello";
  }
}

console.log(MyNamespace.sayHello());
~~~
A. `Hello`

B. `undefined`

C. `null`

D. `Error`

**Answer:** A. `Hello`

### 27. Conditional Types
#### Q27. What does the following code print?
```typescript
type IsString<T> = T extends string ? "yes" : "no";
type Result = IsString<string>;
console.log(Result);
~~~
A. `yes`

B. `no`

C. `undefined`

D. `Error`

Answer: C. `undefined` (TypeScript types are erased at runtime)

28. Mapped Types

Q28. What will be the output of the following code?

```typescript
type Keys = "option1" | "option2";
type Flags = { [K in Keys]: boolean };

let flags: Flags = {
  option1: true,
  option2: false,
};

console.log(flags.option1, flags.option2);
```

A. `true false`

B. `false true`

C. `true true`

D. `false false`

Answer: A. `true false`

29. Decorators

Q29. What is the output of this code snippet?

```typescript
function sealed(target: Function) {
  Object.seal(target);
  Object.seal(target.prototype);
}

@sealed
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
```

  greet() {
    return `Hello, ${this.greeting}`;
  }
}

let greeter = new Greeter("world");
console.log(greeter.greet());
~~~
A. `Hello, world`
B. `undefined`
C. `null`
D. `Error`

**Answer:** A. `Hello, world`

### 30. Index Signatures
#### Q30. What does the following code print?
```typescript
interface StringArray {
  [index: number]: string;
}
~~~

let myArray: StringArray = ["Alice", "Bob"];
console.log(myArray[0]);
~~~
A. `Alice`
B. `Bob`
C. `undefined`
D. `Error`

**Answer:** A. `Alice`

### 31. Utility Types - Partial
#### Q31. What is the output of this code snippet?
```typescript
interface Todo {
  title: string;
  description: string;
}

function updateTodo(todo: Todo, fieldsToUpdate: Partial<Todo>) {
~~~

```
  return { ...todo, ...fieldsToUpdate };
}

const todo1 = { title: "organize desk", description: "clear clutter" };
const todo2 = updateTodo(todo1, { description: "throw out trash" });

console.log(todo2);
```

A. `{ title: "organize desk", description: "throw out trash" }`

B. `{ title: "organize desk", description: "clear clutter" }`

C. `undefined`

D. `Error`

Answer: A. `{ title: "organize desk", description: "throw out trash" }`

32. Utility Types - Readonly

Q32. What does the following code print?

```typescript
interface Todo {
  title: string;
}

const todo: Readonly<Todo> = { title: "Delete inactive users" };
todo.title = "Hello";
console.log(todo.title);
```
A. `Delete inactive users`

B. `Hello`

C. `undefined`

D. `Error`

Answer: D. `Error`

33. Utility Types - Pick
Q33. What is the output of this code snippet?
```typescript
interface Todo {
  title: string;
  description: string;
  completed: boolean;
}

type TodoPreview = Pick<Todo, "title" |
"completed">;

const todo: TodoPreview = {
  title: "Clean room",
  completed: false,
};

console.log(todo);
```

A. `{ title: "Clean room", completed: false }`

B. `{ title: "Clean room", description: undefined, completed: false }`

C. `undefined`

D. `Error`

Answer: A. `{ title: "Clean room", completed: false }`

34. Utility Types - Omit

Q34. What does the following code print?

```typescript
interface Todo {
  title: string;
  description: string;
  completed: boolean;
}

type TodoPreview = Omit<Todo, "description">;

const todo: TodoPreview = {
  title: "Clean room",
  completed: false,
};

console.log(todo);
```

A. `{ title: "Clean room", completed: false }`

B. `{ title: "Clean room", description: undefined, completed: false }`

C. `undefined`

D. `Error`

Answer: A. `{ title: "Clean room", completed: false }`

35. Utility Types - ReturnType

Q35. What is the output of this code snippet?

```typescript
function getUser() {
  return { name: "Alice", age: 25 };
}

type User = ReturnType<typeof getUser>;

let user: User = { name: "Alice", age: 25 };
console.log(user);
```

A. `{ name: "Alice", age: 25 }`

B. `{ name: "Alice" }`

C. `{ age: 25 }`

D. `undefined`

Answer: A. `{ name: "Alice", age: 25 }`

36. Utility Types - Parameters
Q36. What does the following code print?
```typescript
function greet(name: string, age: number) {
  return `Hello ${name}, you are ${age} years old.`;
}

type GreetParameters = Parameters<typeof greet>;

console.log(GreetParameters);
```
A. `[string, number]`

B. `["name", "age"]`

C. `undefined`

D. `Error`

Answer: A. `[string, number]`

37. Utility Types - InstanceType
Q37. What is the output of this code snippet?
```typescript
class User {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
}

type UserType = InstanceType<typeof User>;

let user: UserType = new User("Alice");
console.log(user.name);
```
A. `Alice`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Alice`

38. Utility Types - NonNullable

Q38. What does the following code print?

```typescript
type T = string | null | undefined;
type NonNullableT = NonNullable<T>;

let value: NonNullableT = "Hello";
console.log(value);
```

A. `Hello`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Hello`

39. Utility Types - Extract

Q39. What is the output of this code snippet?

```typescript
type T = string | number | boolean;
type StringType = Extract<T, string>;

let value: StringType = "Hello";
console.log(value);
```

A. `Hello`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Hello`

40. Utility Types - Exclude

Q40. What does the following code print?

```typescript
type T = string | number | boolean;
type NonBoolean = Exclude<T, boolean>;
```

let value: NonBoolean = "Hello";
console.log(value);
~~~

A. `Hello`

B. `undefined`

C. `null`

D. `Error`

Answer:

A. `Hello`

41. Utility Types - Record
Q41. What is the output of this code snippet?
```typescript
type Page = "home" | "about" | "contact";

type PageInfo = Record<Page, { title: string }>;

const pageInfo: PageInfo = {
~~~

  home: { title: "Home Page" },
  about: { title: "About Us" },
  contact: { title: "Contact Us" },
};

console.log(pageInfo.home.title);
~~~
A. `Home Page`

B. `About Us`

C. `Contact Us`

D. `undefined`

**Answer:** A. `Home Page`

### 42. Type Predicates
#### Q42. What does the following code print?
```typescript
function isString(value: any): value is string {
  return typeof value === "string";
}
~~~

let value: any = "Hello";
if (isString(value)) {
  console.log(value.length);
}

A. `5`

B. `undefined`

C. `null`

D. `Error`

**Answer:** A. `5`

### 43. Type Aliases
#### Q43. What is the output of this code snippet?
```typescript
type StringOrNumber = string | number;

let value: StringOrNumber = 42;
console.log(value);
```

A. `42`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `42`

44. Keyof Operator
Q44. What does the following code print?
```typescript
interface Person {
  name: string;
  age: number;
}

type PersonKeys = keyof Person;

let key: PersonKeys = "name";
console.log(key);
```
A. `name`

B. `age`

C. `undefined`

D. `Error`

Answer: A. `name`

45. typeof Operator
Q45. What is the output of this code snippet?
```typescript
let user = { name: "Alice", age: 25 };
type UserType = typeof user;

let newUser: UserType = { name: "Bob", age: 30 };
console.log(newUser);
```
A. `{ name: "Bob", age: 30 }`

B. `{ name: "Alice", age: 25 }`

C. `undefined`

D. `Error`

Answer: A. `{ name: "Bob", age: 30 }`

46. Type Inference

Q46. What does the following code print?

```typescript
let message = "Hello, World!";
console.log(typeof message);
```

A. `string`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `string`

47. Index Signatures with Multiple Types

Q47. What is the output of this code snippet?

```typescript
interface StringOrNumberDictionary {
  [key: string]: string | number;
}

let dictionary: StringOrNumberDictionary = {
```

  name: "Alice",
  age: 30,
};

console.log(dictionary.name, dictionary.age);
~~~
A. `Alice 30`
B. `30 Alice`
C. `undefined`
D. `Error`

**Answer:** A. `Alice 30`

### 48. TypeScript with DOM
#### Q48. What does the following code print?
```typescript
let element = document.createElement("div");
element.textContent = "Hello, World!";
document.body.appendChild(element);
console.log(element.textContent);
~~~

A. `Hello, World!`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Hello, World!`

49. Function Overloads

Q49. What is the output of this code snippet?

```typescript
function add(a: number, b: number): number;
function add(a: string, b: string): string;
function add(a: any, b: any): any {
  return a + b;
}

console.log(add(1, 2));
console.log(add("Hello, ", "World!"));
```

A. `3 Hello, World!`

B. `1,2 Hello, ,World!`

C. `undefined`

D. `Error`

Answer: A. `3 Hello, World!`

50. TypeScript with JSON
Q50. What does the following code print?
```typescript
let jsonString = '{"name": "Alice", "age": 25}';
let user = JSON.parse(jsonString);
console.log(user.name, user.age);
```
A. `Alice 25`

B. `undefined`

C. `null`

D. `Error`

Answer: A. `Alice 25`