TypeScript Advanced Interview Questions (تصادفی و چالش‌برانگیز)

سوال 1: پیاده‌سازی یک `Discriminated Union` با وضعیت‌های `success`، `error`، `loading` و یک تابع که پوشش کامل حالت‌ها را بررسی کند.

پاسخ:

```ts
type Success = { status: 'success'; data: string };
type Failure = { status: 'error'; message: string };
type Loading = { status: 'loading' };
type APIResponse = Success | Failure | Loading;

function handle(res: APIResponse) {
 switch(res.status) {
  case 'success': return res.data.toUpperCase();
  case 'error': return `ERR: ${res.message}`;
  case 'loading': return 'Loading...';
  default:
   const _exhaustive: never = res;
   return _exhaustive;
 }
}
```

توضیح: از `never` برای اطمینان از پوشش کامل استفاده شده است.

سوال 2: یک تابع `fail` بساز که هیچ‌وقت باز نگردد و نوع آن `never` باشد.

پاسخ:

```ts
function fail(msg: string): never {
 throw new Error(msg);
}
```

توضیح: `never` برای توابعی که یا استثنا می‌اندازند یا بی‌نهایت loop دارند استفاده می‌شود.

سوال 3: یک Union و Intersection با `Admin` و `Member` بساز و نحوه تفاوت آنها را توضیح بده.

پاسخ:

```ts
type Admin = { role: 'admin'; permissions: string[] };
type Member = { role: 'member'; points: number };

// Union
```

```ts
type Person = Admin | Member;
const p: Person = { role: 'admin', permissions: ['read'] };

// Intersection
type WithId = { id: string };
type WithTimestamps = { createdAt: Date; updatedAt: Date };
type Entity = WithId & WithTimestamps;
const e: Entity = { id: '1', createdAt: new Date(), updatedAt: new Date() };
```

توضیح: Union یکی از نوع‌ها را می‌پذیرد، Intersection ترکیبی از همه را.

سوال 4: یک Type Guard بساز که بررسی کند یک شیء از کلاس `Dog` است یا نه.

پاسخ:

```ts
class Dog { bark() { console.log('woof'); } }
class Cat { meow() { console.log('meow'); } }

function isDog(x: unknown): x is Dog {
 return x instanceof Dog;
}
```

توضیح: Type Predicate (`x is Dog`) به TypeScript کمک می‌کند نوع متغیر را محدود کند.

سوال 5: یک تابع `assertHas` بساز که بررسی کند یک کلید در یک شیء وجود دارد و در غیر اینصورت خطا دهد.

پاسخ:

```ts
function assertHas<K extends PropertyKey>(obj: unknown, key: K): asserts obj is
Record<K, unknown> {
 if (typeof obj !== 'object' || obj === null || !(key in obj))
  throw new Error('Missing key');
}
```

توضیح: `asserts obj is ...` برای اطمینان TypeScript از پیش‌شرط‌ها استفاده می‌شود.

سوال 6: یک `infer` برای استخراج نوع خروجی تابع بساز.

پاسخ:

```ts
type MyReturnType<F> = F extends (...args: any[]) => infer R ? R : never;
function toPair(n: number) { return [n, String(n)] as const; }
type Pair = MyReturnType<typeof toPair>; // readonly [number, string]
```

توضیح: `infer` اجازه می‌دهد نوع میانی استخراج شود.

سوال 7: یک TypeScript Template Literal Type بساز که ترکیب اندازه و رنگ دکمه را نشان دهد.

پاسخ:

```ts
type Size = 'sm' | 'md' | 'lg';
type Color = 'red' | 'blue' | 'green';
type ButtonVariant = `${Size}-${Color}`;
const v: ButtonVariant = 'sm-red';
```

توضیح: این امکان را می‌دهد که رشته‌ها نوع‌دار باشند و خطاهای احتمالی در زمان کامپایل مشخص شود.

سوال 8: یک تابع curry بنویس که یک تابع دو پارامتری را به توالی توابع یک‌پارامتری تبدیل کند.

پاسخ:

```ts
function curry<A, B, R>(fn: (a: A, b: B) => R) {
 return (a: A) => (b: B) => fn(a, b);
}
```

توضیح: توابع curry باعث انعطاف‌پذیری در فراخوانی توابع می‌شوند.

سوال 9: یک مثال از استفاده `typeof` و `instanceof` برای نوع‌بندی متغیر بساز.

پاسخ:

```ts
function printId(id: string | number) {
 if (typeof id === 'string') console.log(id.toUpperCase());
 else console.log(id.toFixed(2));
}
```

توضیح: `typeof` برای نوع‌های پایه و `instanceof` برای نمونه‌سازی کلاس‌ها استفاده می‌شود.

سوال 10: یک Intersection بساز که ترکیب Admin و Member باشد و role باید سازگار باشد.

پاسخ:

```ts
type AdminCompat = { role: string; permissions: string[] };
type MemberCompat = { role: string; points: number };
type SuperUser = AdminCompat & MemberCompat;
const su: SuperUser = { role: 'admin', permissions: ['*'], points: 100 };
```

توضیح: Intersection نیاز به سازگاری فیلدهای مشترک دارد.

سوال 11: یک تابع `move` بساز که بررسی کند شیء `Fish` است یا `Bird` و تابع مناسب را صدا بزند.

پاسخ:

```ts
type Fish = { swim: () => void };
type Bird = { fly: () => void };
function move(pet: Fish | Bird) {
  'swim' in pet ? pet.swim() : pet.fly();
}
```

توضیح: `in`-guard برای بررسی وجود فیلد در نوع Union استفاده می‌شود.

سوال 12: یک Tuple بساز و از `Head` و `Tail` برای استخراج عناصر استفاده کن.

پاسخ:

```ts
type Head<T extends any[]> = T extends [infer H, ...any[]] ? H : never;
type Tail<T extends any[]> = T extends [any, ...infer R] ? R : never;
type Example = [1, 2, 3];
type H = Head<Example>; // 1
type T = Tail<Example>; // [2,3]
```

توضیح: infer در Tuple برای استخراج عناصر مفید است.

سوال 13: یک نوع EventName بساز که پیشوند `on` و Capitalize روی نام رو اعمال کند.

پاسخ:

```ts
type EventName<K extends string> = `on${Capitalize<K>}`;
type DOMEvents = EventName<'click' | 'focus' | 'blur'>; // 'onClick' | 'onFocus'
| 'onBlur'
```

توضیح: Template Literal Types با قابلیت Capitalize کاربرد دارد برای نام‌گذاری ایونت‌ها.

سوال 14: یک تابع `area` بنویس که از Union Shape استفاده کند و exhaustiveness checking داشته باشد.

پاسخ:

```
type Shape = { kind: 'circle'; r: number } | { kind: 'square'; s: number };
function area(shape: Shape): number {
 if (shape.kind === 'circle') return Math.PI * shape.r ** 2;
 if (shape.kind === 'square') return shape.s ** 2;
 const _x: never = shape;
 return _x;
}
```

توضیح: با استفاده از `never` می‌توان نوع‌های جدید اضافه شده را