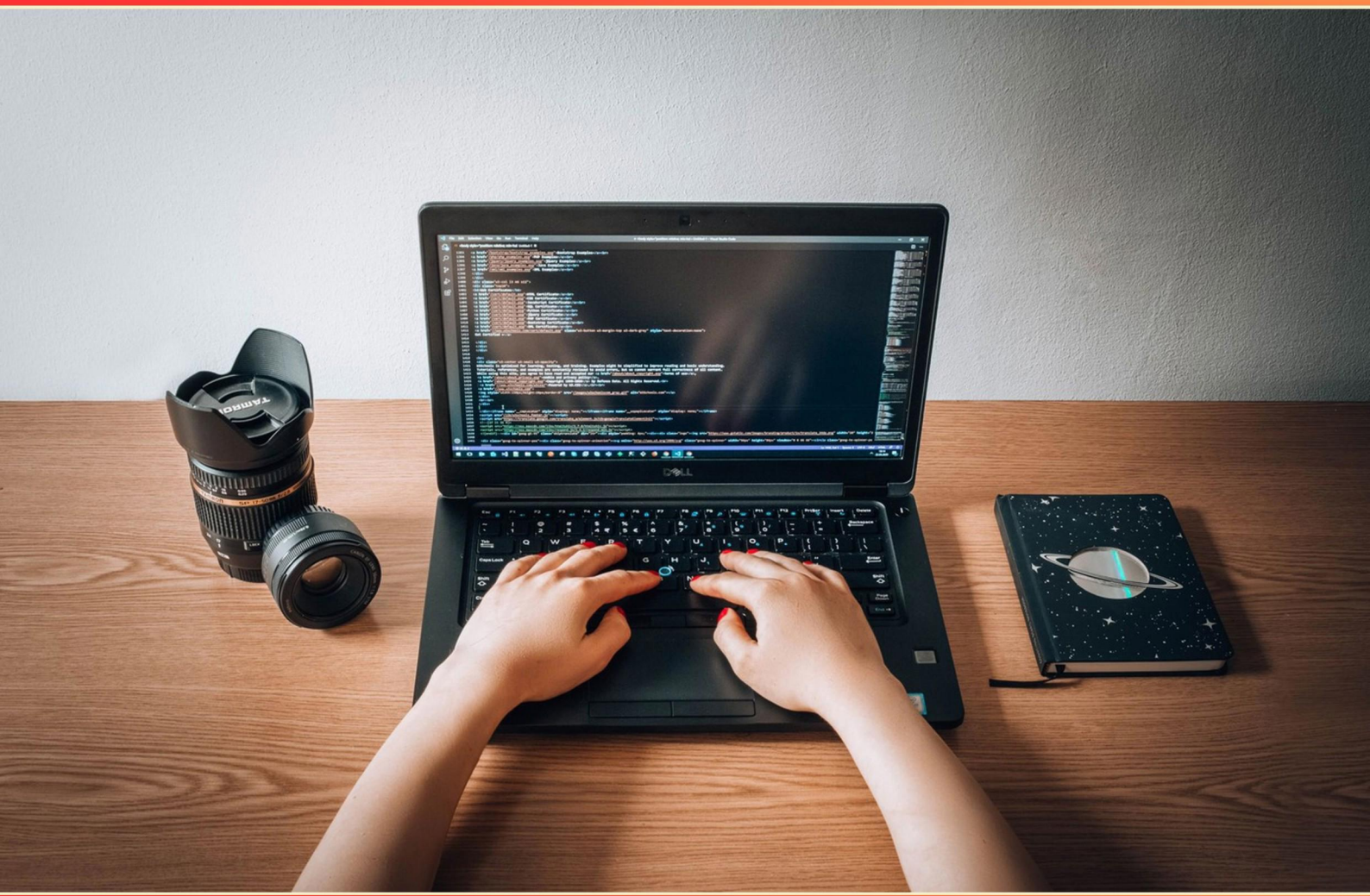
ANGULAR FUNDAMENTALS PRACTICE TEST (SAMPLE)

STUDY GUIDE



SAMPLE STUDY GUIDE WITH LIMITED QUESTIONS

VISIT US ONLINE FOR THE FULL VERSION STUDY GUIDE

https://angularfundamentals.examzify.com

Copyright © 2026 by Examzify - A Kaluba Technologies Inc. product.

ALL RIGHTS RESERVED.

No part of this book may be reproduced or transferred in any form or by any means, graphic, electronic, or mechanical, including photocopying, recording, web distribution, taping, or by any information storage retrieval system, without the written permission of the author.

Notice: Examzify makes every reasonable effort to obtain accurate, complete, and timely information about this product from reliable sources.

SAMPLE

Table of Contents

Copyright 1

Table of Contents 2

Introduction 3

How to Use This Guide 4

Questions 5

Answers 8

Explanations 10

Next Steps 15

SAMPLE

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- Practice answering questions under realistic conditions,
- Improve accuracy and speed,
- Review explanations to strengthen weak areas, and
- Approach the exam with greater confidence.

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

SAMPLE

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here's how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don't just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There's no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

SAMPLE

Questions

SAMPLE

1. Which statement describes how to access a DOM element in a component class using a template reference variable?

A. Use @ViewChild with a selector matching the template reference variable and ElementRef.

B. Use ActivatedRoute to get DOM elements.

C. Use @Output to access DOM elements.

D. Use @ContentChild to access DOM elements.

2. What is the role of an Angular module (NgModule) and which metadata properties are commonly defined on it?

A. NgModule groups related components, directives, pipes, and services; key metadata: declarations, imports, providers, bootstrap, and exports.

B. NgModule is a UI control that renders templates.

C. NgModule is a directive that modifies the DOM.

D. NgModule handles routing automatically.

3. Standalone components enable easier lazy loading.

A. They require NgModules for lazy loading.

B. They enable easier lazy loading.

C. They prevent lazy loading entirely.

D. They have no impact on lazy loading.

4. Which approach primarily improves initial load time by loading parts of the application only when they are needed?

A. Lazy loading

B. Differential loading

C. Tree shaking

D. Hydration

5. Which decorators are used to access child components or content?

A. @ViewChild accesses child components/elements in the view; @ContentChild accesses projected content.

B. @ContentChild accesses services; @ViewChild accesses routing.

C. @ViewChild and @ContentChild are used to define routes.

D. @ViewChild accesses DOM events; @ContentChild accesses HTML templates.

SAMPLE

6. How do you navigate programmatically within a component?

A. Inject Router and call navigate or navigateByUrl

B. Use window.location to change pages

C. Use a link with routerLink to perform navigation

D. Use ActivatedRoute to navigate

7. Which combination reduces the initial bundle size in Angular?

A. Lazy load features, remove unused deps, optimize assets, and enable tree shaking.

B. Increase eager imports in AppModule.

C. Disable minification to simplify debugging.

D. Bundle everything into a single file.

8. What is a route resolver used for?

A. Pre-fetches data before route activation to ensure data availability

B. Handles navigation events globally

C. Configures route reuse strategies

D. Provides authentication tokens

9. Under what circumstance would you choose to use an InjectionToken?

A. When injecting a non-class value or interface.

B. When injecting a component.

C. When injecting a service by type.

D. When injecting a module.

10. What is an HTTP interceptor and typical use cases?

A. Interceptor handles routing guard logic

B. Interceptor patches user data in local storage

C. Interceptor manages template rendering

D. Interceptor intercepts requests/responses to add headers, log activity, handle errors, or show loaders

SAMPLE

Answers

SAMPLE

1. A
2. A
3. B
4. A
5. A
6. A
7. A
8. A
9. A
10. D

SAMPLE

Explanations

SAMPLE

1. Which statement describes how to access a DOM element in a component class using a template reference variable?

A. Use @ViewChild with a selector matching the template reference variable and ElementRef.

B. Use ActivatedRoute to get DOM elements.

C. Use @Output to access DOM elements.

D. Use @ContentChild to access DOM elements.

Accessing a DOM element from a component class using a template reference variable relies on Angular's view querying. In the template you create a reference like <div #myElem>, and in the component you use the ViewChild decorator with that reference name and an ElementRef to get a handle to the actual DOM element. For example, @ViewChild('myElem') elem!: ElementRef; then after the view initializes you can access the element via this.elem.nativeElement. This approach is the correct one because it directly queries the component's own view for the element tied to the template reference variable and provides a native DOM handle through ElementRef. Other options don't fit: ActivatedRoute is for navigation and routing data, not DOM elements; @Output is for event emission to a parent component; and @ContentChild is for elements projected into the component via ng-content, not elements inside the component's own template.

2. What is the role of an Angular module (NgModule) and which metadata properties are commonly defined on it?

A. NgModule groups related components, directives, pipes, and services; key metadata: declarations, imports, providers, bootstrap, and exports.

B. NgModule is a UI control that renders templates.

C. NgModule is a directive that modifies the DOM.

D. NgModule handles routing automatically.

An NgModule defines a boundary and context for a piece of an Angular app. It groups related components, directives, pipes, and services and tells Angular how to compile, what it depends on, and what it shares with the rest of the app. The common metadata you'll define on an NgModule includes: - declarations: the components, directives, and pipes that belong to this module. These are the building blocks that Angular compiles within the module's scope. - imports: other modules whose exported declarations are needed by the templates in this module. This is how you bring in functionality from Angular itself (like CommonModule) or shared feature modules. - providers: services that should be available for dependency injection within this module (and potentially to its child modules). This configures the injector with the module's services. - bootstrap: the root component to bootstrap when starting the application; this is used only in the root AppModule to kick off the app. - exports: a subset of declarations that should be visible to other modules that import this module, effectively making them public. These properties together define what the module contains, what it needs, what it provides via DI, and what it exposes to the rest of the application.

SAMPLE

3. Standalone components enable easier lazy loading.

A. They require NgModules for lazy loading.

B. They enable easier lazy loading.

C. They prevent lazy loading entirely.

D. They have no impact on lazy loading.

Standalone components simplify lazy loading by removing the need to wrap a piece of UI in an NgModule before it can be lazy-loaded. In Angular, traditionally lazy loading depended on NgModules to declare and export the pieces that could be loaded on demand. Standalone components are self-contained: they declare their own dependencies in an imports array and are designated as standalone. This lets the router load a component directly (for example, via a loadComponent route) without requiring a separate feature module. As a result, code-splitting and on-demand loading become simpler and more modular, since each component can be lazy-loaded independently and with less boilerplate. So the statement that standalone components enable easier lazy loading is correct.

4. Which approach primarily improves initial load time by loading parts of the application only when they are needed?

A. Lazy loading

B. Differential loading

C. Tree shaking

D. Hydration

Lazy loading loads parts of the application only when they're actually needed, which trims the initial bundle size and speeds up the first render. By splitting the app into separate chunks and fetching them on demand (often via dynamic imports or router configuration), the browser downloads less JavaScript upfront and only retrieves additional code as the user navigates to new routes or features. Differential loading creates different bundles for different environments (like older vs newer browsers) to optimize payloads, but it's not about delaying loading of parts of the app. Tree shaking removes unused code at build time to shrink the bundle, yet that happens before the app runs rather than delaying loading during navigation. Hydration is the process of attaching client-side behavior to server-rendered HTML, not about loading on demand. For Angular, this approach is commonly implemented with lazy-loaded modules that load via the router when a specific route is accessed.

5. Which decorators are used to access child components or content?

A. @ViewChild accesses child components/elements in the view; @ContentChild accesses projected content.

B. @ContentChild accesses services; @ViewChild accesses routing.

C. @ViewChild and @ContentChild are used to define routes.

D. @ViewChild accesses DOM events; @ContentChild accesses HTML templates.

Think of how components manage their own template versus what a parent passes in. In Angular, two decorators are used to grab references to parts of a component: ViewChild looks inside the component's own template to get a reference to a child component, directive, or DOM element so you can access its properties or call its methods. ContentChild looks into the content that is projected into the component with ng-content, letting you interact with those projected parts from the host component. This distinction matters because you sometimes need to work with what's declared in the component's template and other times with what a parent projects into it. You usually access these references after rendering, using AfterViewInit or AfterContentInit. The other options don't fit: one misstates that ContentChild is for services or that ViewChild is for routing; another suggests these decorators define routes; and another implies they deal with DOM events or HTML templates. In short, ViewChild is for view-contained elements and components, while ContentChild is for projected content.

SAMPLE

6. How do you navigate programmatically within a component?

A. Inject Router and call navigate or navigateByUrl

B. Use window.location to change pages

C. Use a link with routerLink to perform navigation

D. Use ActivatedRoute to navigate

Programmatic navigation in Angular is done by using the Router service. By injecting Router into your component and calling its methods, you can navigate from code without causing a full page reload. Use this.router.navigate([...commands]) for relative or complex routes, or this.router.navigateByUrl('/path') when you have a specific URL to go to. This approach keeps navigation within the Angular router so guards, resolvers, and animations work as configured, and it preserves the single-page application behavior. Using window.location to change pages causes a full page reload and bypasses the Angular router, breaking SPA benefits. A template-driven link with routerLink handles navigation declaratively in the HTML but isn't the programmatic approach you need inside component logic. ActivatedRoute is about reading information about the current route (parameters, query params, etc.), not initiating navigation.

7. Which combination reduces the initial bundle size in Angular?

A. Lazy load features, remove unused deps, optimize assets, and enable tree shaking.

B. Increase eager imports in AppModule.

C. Disable minification to simplify debugging.

D. Bundle everything into a single file.

Reducing the initial bundle size comes from loading only what's essential up front and shrinking what is included in that first download. Lazy loading features is key here because it splits the app into multiple bundles and defers loading parts of the code until you actually navigate to the routes that require them. That means the initial bundle contains just the core essentials, not every feature you might use later. Removing unused dependencies helps prevent dead code from being pulled into the bundle. When you keep libraries or modules that aren't actually used, they end up increasing the size of what loads initially, even if they aren't needed right away. Optimizing assets also contributes to a smaller initial payload. Compressed images, fonts, and other assets reduce the total data the user must fetch on startup, making the initial load feel faster even when the JavaScript bundle itself isn't dramatically changed. Enabling tree shaking ensures that only the code actually referenced by your app is kept during the build. Unused exports from libraries get dropped, so the final bundle doesn't carry unnecessary functions or modules, further shrinking the initial size. Together, these approaches minimize the amount of code and data delivered before the app becomes interactive, which is why this combination is the best choice. In contrast, increasing eager imports would blow up the initial bundle, disabling minification would keep the bundle larger and harder to debug, and bundling everything into a single file would defeat the purpose of splitting code and assets.

SAMPLE

8. What is a route resolver used for?

A. Pre-fetches data before route activation to ensure data availability

B. Handles navigation events globally

C. Configures route reuse strategies

D. Provides authentication tokens

A route resolver fetches data before the route activates so the component has what it needs as soon as it is created. In Angular, a resolver implements Resolve<T> and is connected to a route via the resolve configuration. The router waits for the resolver to complete (returning data via an observable or promise) and only then activates the route and instantiates the component. This means you can access the resolved data immediately from ActivatedRoute.data, avoiding partial rendering or extra loading within the component itself. If data loading fails, you can handle it in the resolver or with guards to navigate elsewhere or show an error. Other router features operate in different ways: navigation events are emitted by the Router as it moves, route reuse strategies control how components are kept or replaced during navigation, and tokens are typically provided by authentication mechanisms like guards or HTTP interceptors.

9. Under what circumstance would you choose to use an InjectionToken?

A. When injecting a non-class value or interface.

B. When injecting a component.

C. When injecting a service by type.

D. When injecting a module.

Injection tokens are used when you want to inject something that isn't a class, such as a configuration object, a string, or a value that represents an interface type at compile time but doesn't exist as a runtime class. Angular's dependency injection uses a token to look up the provider; for classes, the class type itself serves as the token. But interfaces disappear at runtime, so they cannot be used as tokens. By creating an InjectionToken<T>, you establish a unique, type-safe token to map to a value. For example, you can define a token for a config value and provide a concrete object for it, then inject that value in a class using @Inject(token). This keeps the code strongly typed while supplying non-class data. The same pattern works for interface-like shapes: define an InjectionToken<MyInterface>, provide a concrete object under that token, and inject it with @Inject(theToken).

10. What is an HTTP interceptor and typical use cases?

A. Interceptor handles routing guard logic

B. Interceptor patches user data in local storage

C. Interceptor manages template rendering

D. Interceptor intercepts requests/responses to add headers, log activity, handle errors, or show loaders

Requests and responses pass through an HTTP interceptor, which sits in the HTTP pipeline and lets you handle all HTTP traffic in one place. An interceptor hooks into the request/response cycle, so you can inspect and modify outgoing requests and process incoming responses before your code uses them. In practice, you typically clone the request to add things like authorization headers, then pass it along. You can also tap into the response to log activity, implement centralized error handling, transform data, or show and hide a loading indicator. This approach keeps cross-cutting concerns consistent across all HTTP calls instead of duplicating logic in each service. Other descriptions don't fit the role: routing guards are separate mechanisms for protecting routes, template rendering is about the view layer, and patching local storage isn't the interceptor's primary function.

SAMPLE

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://angularfundamentals.examzify.com

We wish you the very best on your exam journey. You've got this!

SAMPLE