PDI Training Course

Platform Developer I

Structured Learning & Certification Preparation

Table of Contents

Introduction

The Platform Developer I (PDI) certification from Salesforce validates the foundational ability to develop custom applications on the Salesforce platform using both declarative configuration and programmatic development. It confirms that a candidate understands how to extend platform functionality through code, automation, and user interface customization while working within the platform's architectural principles. In modern cloud environments where organizations rely on highly configurable platforms, this certification represents an important baseline for developers building scalable business solutions.

About This Training / Certification

The Platform Developer I certification evaluates the ability to design and implement solutions using the development tools available within the Salesforce platform. It focuses on core programming concepts, platform architecture, data modeling, and application logic development. Candidates are expected to understand how declarative automation tools interact with custom code and how applications are structured within the platform ecosystem.

This certification is generally positioned at the foundational to early-intermediate level for Salesforce development professionals. It is commonly pursued by individuals transitioning from administrative roles into development responsibilities, as well as by developers who want to demonstrate competence in building and customizing applications on the Salesforce platform as part of a broader Salesforce technical career path.

What We Offer (AAAdemy)

AAAdemy provides structured training resources designed to support certification preparation and skill development across a wide range of IT domains. Our learning materials are built around clear knowledge structures, practical study guidance, and exam-oriented practice to help learners progress with confidence.

We offer well-organized knowledge explanations that break down complex topics into clear, understandable sections aligned with official exam objectives and real-world skill requirements. Each topic is designed to support both conceptual understanding and practical application.

Our study plans and learning guidance help learners follow a logical progression, focusing on key concepts, common pitfalls, and effective preparation strategies. This approach enables learners to study efficiently while maintaining a clear view of their learning goals.

To reinforce understanding, AAAdemy also provides practice questions and exam-focused insights that reflect typical certification scenarios. These resources are intended to help learners evaluate their readiness and strengthen their confidence before taking an exam.

All content is designed for flexible, self-paced learning, allowing individuals to study independently or alongside their existing professional or academic commitments.

Knowledge Overview

The Platform Developer I certification focuses on several key knowledge areas that represent the essential components of application development on the Salesforce platform.

Developer Fundamentals Area
This area focuses on the core architectural principles of the Salesforce platform. Candidates are expected to understand the multitenant environment, the platform's data architecture, and how custom applications are structured. Knowledge includes understanding objects, fields, relationships, metadata-driven development, and how platform features support scalable application design.

Process Automation and Logic Area
This area examines how business logic and automation are implemented within the platform. Candidates should understand when to use declarative automation tools and when programmatic approaches are appropriate. Concepts include implementing automated processes, applying conditional logic, and designing maintainable automation solutions that interact with platform data and workflows.

User Interface Area
The user interface area focuses on building and customizing application interfaces that allow users to interact with platform data. Candidates should understand how platform components support application usability and how developers can extend user interfaces to create responsive and functional experiences for business users.

Testing, Debugging, and Deployment Area
This domain emphasizes the importance of validating and maintaining application functionality. Candidates should understand testing principles, debugging techniques, and the role of testing in ensuring system reliability. Knowledge also includes understanding how development artifacts are managed and moved across environments while maintaining stability and quality.

Detailed Knowledge Explanation

PDI Developer Fundamentals

1. Data Modeling and Management

1.1. The foundation of any Salesforce application is its data model, where objects serve as the structural equivalent of database tables to organize records and fields. Standard Objects, including Account, Contact, Opportunity, and Case, provide a robust, predefined framework for common business entities. While these objects are non-deletable, they are highly extensible. Custom Objects, identified by the "__c" suffix in the API, allow architects to build proprietary data structures tailored to unique organizational requirements. The Schema Builder provides the necessary visual interface for managing this architecture, offering a drag-and-drop environment to define objects and visualize their complex interconnections.

1.2. Fields define the discrete data points within an object and are categorized to ensure data validity and system performance. Scalar fields handle basic data types such as Text, Number, and Date. Formula fields provide read-only logic that automatically calculates values based on other record data. Relationship fields are critical for data integrity, spanning Lookup relationships for loose connectivity where the field may remain null, and Master-Detail relationships for high-dependency links where the child record’s lifecycle is strictly tied to the parent. Many-to-Many relationships are implemented via junction objects, enabling complex many-to-one mapping between two distinct objects.

1.3. A Senior Developer’s design strategy must prioritize data integrity and system maintainability by selecting the most restrictive and accurate field types. For instance, utilizing Picklists instead of open Text fields is a vital design tip to ensure data consistency for reporting and downstream automation. Furthermore, numeric data must be stored in Number fields to support mathematical operations. Architects must also limit the depth of Master-Detail relationships to prevent unnecessary complexity and performance degradation. Choosing the correct field type at the outset is the "So What?" of data modeling, as it dictates the scalability of the entire application.

2. Apex Programming Language

2.1. Apex is a proprietary, strongly-typed, object-oriented language specifically engineered to execute within the multi-tenant constraints of the Salesforce platform. While it shares a familiar syntax with Java, making it an ideal transition for object-oriented developers, Apex is uniquely specialized for interacting with the Salesforce database. The language utilizes standard control structures like if-else statements and loops, alongside robust data structures including Lists for ordered data, Sets for unique values, and Maps for efficient key-value pair mapping.

2.2. The structural maturity of Apex is realized through classes and methods, where encapsulation is strictly managed via access modifiers. Developers use the private modifier for internal logic, public for access within the same namespace, and global for code that must be accessible across all namespaces. Static methods are frequently utilized for utility logic, as they can be invoked directly on the class without requiring an instance, facilitating cleaner and more modular code.

2.3. Triggers represent the programmatic gateway for automation, executing logic in response to database events. To maintain a performant environment, a Senior Lead enforces the "one trigger per object" rule to centralize execution order and prevent logic conflicts. Strategic use of static variables is also required to manage execution state and prevent recursive loops. Mastering these programming fundamentals allows developers to transition from basic record manipulation to complex, automated business logic.

3. SOQL and SOSL

3.1. A developer’s toolkit must distinguish between the precision of Salesforce Object Query Language (SOQL) and the broad reach of Salesforce Object Search Language (SOSL) to ensure efficient resource consumption. A SOQL query is structured around the `SELECT` statement, identifying specific fields `FROM` a primary object and optionally filtering results via a `WHERE` clause. This language is the primary tool for targeted data retrieval when the specific object and relationships are known.

3.2. Conversely, SOSL is optimized for full-text search across multiple objects simultaneously. SOSL syntax employs a `FIND` statement to search across indices, specifying search terms `IN ALL FIELDS` and `RETURNING` specific object and field arrays. Choosing SOQL for precise record filtering within Apex logic or SOSL for global search requirements is a strategic decision that directly impacts the speed and efficiency of the application's data layer.

4. Platform Limits (Governor Limits)

4.1. Due to the shared nature of Salesforce’s multi-tenant architecture, Governor Limits are strictly enforced to prevent any single organization from monopolizing underlying hardware resources. These limits act as mandatory guardrails, imposing ceilings such as a maximum of 100 SOQL queries and 150 DML operations per transaction, along with a 10-second limit for synchronous CPU time.

4.2. The "So What?" factor of Governor Limits is the absolute requirement for "bulkified" code. Developers cannot process records individually; instead, all logic must be designed to handle collections of records within a single execution context. Failure to respect these constraints leads to runtime exceptions and system instability. Understanding these limits is the primary driver for choosing advanced architectural patterns like asynchronous processing.

5. Development Tools

5.1. Salesforce provides a tiered toolset to accommodate different phases of the development lifecycle. The Developer Console serves as a browser-based environment for immediate debugging, executing anonymous Apex, and inspecting debug logs. For professional metadata management, the Salesforce CLI offers a robust command-line interface that facilitates automation and integration with external systems.

5.2. Visual Studio Code, integrated with Salesforce extensions, is the modern standard IDE for enterprise development. It supports a source-driven workflow for both Apex and Lightning Web Components, allowing teams to utilize version control and sophisticated debugging tools. These environments provide the necessary infrastructure to transition from simple logic to enterprise-grade architectural patterns.

6. Salesforce MVC Architecture

6.1. The Model-View-Controller (MVC) design pattern is the strategic framework used to ensure separation of concerns within a Salesforce application. The Model, or Data Layer, is defined by the objects, fields, and relationships that represent the business data. The View, or UI Layer, encompasses the components that display this data to the user, such as Lightning Web Components or Visualforce pages. The Controller, or Logic Layer, processes user interactions and enforces business rules through Apex classes and Flows.

6.2. This architecture simplifies maintenance in complex environments by isolating changes. For instance, a Lightning Web Component acts as a View that captures a user's input, which is then passed to a Flow acting as a Controller to execute logic. Because these layers are distinct, a developer can update the UI without disrupting the underlying data model or the core business logic, ensuring a more scalable and resilient application.

7. Apex Security

7.1. Security in Apex is a multi-layered responsibility that requires developers to manually enforce permissions that the system context might otherwise ignore. While sharing rules are governed by the "With Sharing" and "Without Sharing" keywords, developers must programmatically check for Field-Level Security (FLS) and CRUD permissions. This is achieved through specific Apex checks such as `isAccessible` for read permissions and `isCreateable` for create permissions.

7.2. Utilizing "With Sharing" is the professional standard to ensure the code respects the organization's security model. "Without Sharing" should be reserved strictly for system-level operations where unrestricted access is a documented necessity. By manually enforcing these checks, developers prevent unauthorized data access and maintain the integrity of sensitive information within the multi-tenant environment.

8. Asynchronous Apex

8.1. Asynchronous processing is the primary mechanism for managing long-running operations that would otherwise exceed synchronous Governor Limits. Future Methods are suitable for lightweight tasks such as simple external API callouts, whereas Queueable Apex offers a more modern approach that supports complex object types and method chaining. Batch Apex is the essential tool for processing large datasets of up to millions of records by breaking them into manageable chunks.

8.2. The strategic value of asynchronous Apex lies in its ability to prevent blocking user actions. By moving heavy computations or high-volume data updates to the background, the system maintains a responsive user experience. This approach is not merely an option but a requirement for ensuring platform stability during intensive enterprise operations.

9. Metadata API & Deployment

9.1. The Metadata API is the underlying engine for moving configurations and code between Salesforce environments, operating via SOAP-based calls to manage components like objects and Apex classes. For small-scale deployments, developers may use the GUI-based simplicity of Change Sets. However, for large-scale enterprise projects, the Salesforce CLI (SFDX) is the preferred tool for managing complex, source-driven deployments.

9.2. Implementing a CI/CD pipeline significantly enhances release quality by integrating version control systems like GitHub with automated testing. This ensures that every push of code is validated against the target environment, reducing the risk of manual errors and improving team collaboration. These mechanisms represent the final stage of transitioning programmatic and declarative logic into production.

10. Developer Fundamentals Practice Question

Q1: Which statement is true about Standard and Custom Objects in Salesforce?
A. Standard Objects can be deleted, but Custom Objects cannot.
B. Custom Objects are predefined by Salesforce and cannot be modified.
C. Standard Objects can have Master-Detail relationships, but Custom Objects cannot.
D. Custom Objects can have their own fields, relationships, and layouts.

Q2: What is the main difference between Lookup Relationship and Master-Detail Relationship in Salesforce?
A. Lookup Relationship allows cascading deletes, while Master-Detail does not.
B. Master-Detail Relationship allows sharing settings to be inherited, while Lookup does not.
C. Lookup Relationship requires a related record, while Master-Detail allows null values.
D. Master-Detail Relationship does not support roll-up summary fields.

Q3: Which field type should be used to store a computed value that updates automatically when related fields change?
A. Text
B. Number
C. Formula
D. Checkbox

Q4: What is the correct syntax for writing an SOQL query that retrieves all Account names where Industry is 'Technology'?
A. `SELECT Name FROM Account WHERE Industry = 'Technology'`
B. `SELECT Name, Industry FROM Account WHERE Industry == 'Technology'`
C. `FIND 'Technology' IN Account RETURNING Name`
D. `SELECT Name FROM Account WHERE Industry LIKE '%Technology%'`

Q5: In an Apex Trigger, which object contains the new values of records being inserted?
A. `Trigger.Old`
B. `Trigger.New`
C. `Trigger.NewMap`
D. `Trigger.OldMap`

Q6: What is the primary purpose of Governor Limits in Salesforce?
A. To allow unlimited processing power for developers
B. To prevent a single tenant from consuming excessive shared resources
C. To speed up SOQL queries
D. To enforce strict security policies

Q7: Which of the following is NOT a valid asynchronous Apex method in Salesforce?
A. Future Methods
B. Queueable Apex
C. Batch Apex
D. Trigger Apex

Q8: What is the best way to deploy Apex code from a Developer Sandbox to Production in Salesforce?
A. Directly editing the production code
B. Using Change Sets
C. Using Data Loader
D. Sending the code via email

Q9: In Apex, which keyword is used to ensure that a class follows organization-wide sharing rules?
A. `without sharing`
B. `private sharing`
C. `with sharing`
D. `global sharing`

Q10: In an Apex Test Class, which method should be used to create test data without affecting existing records?
A. `System.insert`
B. `Test.startTest()`
C. `@isTest`
D. `Test.loadData()`

PDI Process Automation and Logic

1. Declarative Automation

1.1. Salesforce's suite of declarative tools provides a "no-code" approach to automation, significantly reducing development time. Process Builder is utilized for straightforward logic such as field updates, record creation, or sending email notifications based on record changes. Flow Builder provides a more advanced environment for building multi-step workflows, dynamic user forms, and complex logic involving loops and data queries.

1.2. Approval Processes automate the formal authorization of business records, such as job offers or discount requests, by defining specific approval steps and actions. The strategic value of these tools lies in their maintainability; however, a Senior Developer must recognize when business requirements exceed declarative capabilities and require a transition to programmatic solutions.

2. Programmatic Automation

2.1. Programmatic automation via Apex Triggers and Asynchronous Tasks is required for complex calculations or high-volume processing that declarative tools cannot handle. Triggers are executed based on context: Before triggers are the professional choice for data validation and auto-populating fields, while After triggers are used for operations that require the record ID, such as updating related records.

2.2. Programmatic logic is the "So What?" for enterprise-scale requirements. It allows for a level of precision and performance that ensures the application can handle intricate data structures and massive datasets without failure. By balancing declarative and programmatic approaches, developers can build an automation strategy that is both efficient and highly scalable.

3. Workflow Rules

Workflow Rules remain relevant as a legacy tool for managing technical debt in older Salesforce instances. While they are being phased out in favor of Flow, they possess the unique capability to send outbound messages without custom code—a feature Process Builder does not natively support. They also handle simple email alerts, field updates, and task assignments. Understanding their historical role is necessary for any developer tasked with maintaining or migrating legacy system logic.

4. Best Practices: Process Builder vs. Flow

Salesforce has officially recommended migrating from Process Builder to Flow, as Flow is inherently more powerful and scalable. Flow should be the primary choice for almost all declarative scenarios, including those requiring user interaction or cross-object logic. Apex should only be utilized when Flow hits its performance limits, such as when executing complex HTTP callouts or requiring high-performance batch processing.

5. Trigger Framework (Best Practices for Apex Triggers)

A Trigger Framework is an architectural necessity to ensure modularity and prevent recursion in complex applications. By using a single trigger per object and delegating logic to a "Handler Class," developers keep code organized and reusable. To solve the specific problem of recursion—such as an update on an Account triggering another update on that same Account—developers must use static variables as flags to track execution state and ensure logic only runs once.

6. Flow vs. Apex – When to Use Each?

The decision between Flow and Apex is a trade-off between the ease of declarative maintenance and the advanced control of programmatic logic. Flow is the preferred tool for standard business rules and approvals. Apex is required when dealing with complex data structures, high-volume batching, or intricate integrations where Flow's declarative nature cannot provide the necessary granular control.

7. Asynchronous Apex – Choosing the Right Approach

Choosing the correct asynchronous method is a matter of technical requirement. Future methods are legacy tools best suited for simple primitives and basic callouts. Queueable Apex is the modern professional standard because it supports complex object types and allows for job chaining. Batch Apex is designed for massive data migrations, while Scheduled Apex is used for recurring system maintenance, such as nightly metric recalculations.

8. Process Automation and Logic Practice Question

Q1: Which declarative automation tool is recommended for creating complex multi-step business processes with user input?
A. Workflow Rules

B. Process Builder
C. Flow Builder
D. Approval Process

Q2: Which of the following is NOT a capability of Workflow Rules?
A. Sending an outbound message
B. Creating a new record
C. Sending an email alert
D. Updating a field

Q3: What is a key advantage of using Flow over Process Builder?
A. Flow supports multiple criteria in one automation
B. Flow allows user input and screens
C. Process Builder is recommended for bulk data updates
D. Process Builder can execute before a record is saved

Q4: In which situation should you use an Approval Process instead of Flow or Process Builder?
A. Sending an automated email when a field changes
B. Automatically updating a record's status
C. Implementing a multi-step manager approval workflow
D. Creating a new related record

Q5: What is the correct way to write a trigger that updates a Contact's Description before inserting it?
A.

```
trigger ContactTrigger on Contact (after insert) {
   for (Contact con : Trigger.new) {
       con.Description = 'Welcome!';
   }
}
```

B.

```
trigger ContactTrigger on Contact (before insert) {
   for (Contact con : Trigger.new) {
       con.Description = 'Welcome!';
   }
}
```

C.

```
trigger ContactTrigger on Contact (before update) {
   for (Contact con : Trigger.new) {
      con.Description = 'Welcome!';
   }
}
```

D.

```
trigger ContactTrigger on Contact (before insert, before update) {
   for (Contact con : Trigger.new) {
      con.Description = 'Welcome!';
   }
}
```

Q6: What is the primary reason for using Trigger Frameworks in Salesforce?
A. To avoid recursive execution of triggers
B. To allow multiple triggers on the same object
C. To execute SOQL queries in bulk
D. To run triggers asynchronously

Q7: Which type of Asynchronous Apex should be used when processing a large number of records in small batches?
A. Future Methods
B. Queueable Apex
C. Batch Apex
D. Scheduled Apex

Q8: What is a key advantage of Queueable Apex over Future Methods?
A. Queueable Apex runs synchronously
B. Queueable Apex allows chaining of jobs
C. Queueable Apex does not require test classes
D. Queueable Apex can process records in batches

Q9: What is the correct way to schedule a Scheduled Apex Job to run daily at 12 PM?
A. System.schedule('Daily Job', '0 0 12 * * ?', new ScheduledJob());

B. System.scheduleJob('Daily Job', '0 0 12 * * ?', new ScheduledJob());

C. new ScheduledJob().schedule('Daily Job', '0 0 12 * * ?');

D. System.scheduleBatch(new ScheduledJob(), 'Daily Job', '0 0 12 * * ?');

PDI Testing, Debugging, and Deployment

1. Unit Testing

1.1. Unit testing is a mandatory safeguard in Salesforce development, with a 75% code coverage minimum required for all production deployments. However, a Technical Lead advocates for a 100% coverage target to ensure total system reliability. Testing involves creating data in a test class annotated with @isTest, invoking the code, and using System.assert to verify that the results match expected outcomes. This process is the ultimate "So What?" for ensuring production stability and preventing regressions.

2. Debugging

2.1. Troubleshooting is primarily conducted through the Developer Console and Debug Logs. By strategically placing System.debug statements within the code, developers can trace logic flow and inspect variable states. Filtering logs by levels—such as Debug, Error, or Event—allows for the rapid identification of defects in a high-volume execution environment.

3. Deployment

3.1. Developers must choose between the GUI-based Change Sets and the automated Salesforce DX (SFDX) for environmental migration. While Change Sets are simple for minor updates, they represent a risk for large-scale enterprise deployments due to their manual nature. SFDX is the professional standard, enabling a source-driven, version-controlled development cycle that supports modern CI/CD practices.

4. @TestSetup Annotation

The @TestSetup annotation provides significant efficiency gains by allowing developers to create shared test data once for all methods in a class. This reduces redundant data creation, resulting in faster test execution and a more consistent data state. Using this annotation is a best practice for optimizing the testing lifecycle and maintaining clean, professional test code.

5. Test.startTest() and Test.stopTest()

These methods are essential for isolating the code under test and controlling the execution context. Test.startTest() resets Governor Limits to provide a fresh context for performance validation, while Test.stopTest() forces any asynchronous processes to complete before assertions are made. This ensures that the results of background operations are fully processed and ready for verification.

6. Debug Log Filtering

Efficient log management is critical when troubleshooting complex transactions. Within the Developer Console, developers should set the Log Level to "Debug" and utilize the "Debug Only" filter. This procedure isolates

relevant System.debug messages and removes system-generated noise, allowing the developer to focus exclusively on the logic path being investigated.

7. Salesforce CLI (SFDX) Deployment

SFDX commands enable a professional, command-line driven development workflow. Commands such as `force:source:push` are used to sync local changes to scratch orgs, while `force:source:deploy` is used for deploying metadata to persistent environments. These commands are the foundation of a modern, automated development cycle that ensures code is versioned and deployable.

8. Deployment Pipeline (CI/CD)

A Salesforce CI/CD pipeline automates the journey from development to production using tools like GitHub for version control and GitHub Actions or Jenkins for automation. This pipeline ensures that every code change is automatically tested and validated before release. This automation is the standard for modern teams, ensuring consistent, error-free deployments and facilitating better collaboration across enterprise projects.

9. Testing, Debugging, and Deployment Practice Question

Q1: What is the minimum code coverage percentage required before deploying Apex code to production in Salesforce?
A. 50%
B. 75%
C. 85%
D. 100%

Q2: What is the purpose of the `@TestSetup` annotation in a test class?
A. To define asynchronous test methods
B. To execute test data creation once for all test methods in a class
C. To indicate that a method contains test assertions
D. To deploy test methods to production

Q3: What is the correct way to execute an asynchronous Apex method in a unit test?
A. Call the method directly in a test method
B. Use `Test.startTest()` and `Test.stopTest()`
C. Use `System.debug()` to check the method execution
D. Use `@future` inside the test method

Q4: What is the best way to view System.debug() output in Salesforce?
A. Open the Debug Logs in Developer Console
B. Check the Apex Class definition
C. Use SOQL to query Debug Logs
D. Debug statements are not visible in Salesforce

Q5: How can you filter logs in the Developer Console to focus only on debug messages?
A. Enable SOQL Analyzer
B. Use "Debug Only" filter in the Logs tab
C. Increase Log Levels to FATAL
D. Use `System.assert()` instead of `System.debug()`

Q6: Which deployment tool allows manual selection of metadata components to migrate between Salesforce orgs?
A. Change Sets
B. Salesforce CLI (SFDX)
C. Workbench
D. SOQL Queries

Q7: Which of the following is NOT true about Change Sets?
A. They can be used to migrate Apex classes, objects, and workflows
B. They allow deployment between unrelated Salesforce orgs
C. They require validation before deployment
D. They do not support all metadata types

Q8: Which Salesforce CLI (SFDX) command is used to deploy metadata to an org?
A. `sfdx force:data:bulk:upsert`
B. `sfdx force:source:push`
C. `sfdx force:auth:web:login`
D. `sfdx force:org:open`

Q9: What is the primary benefit of using CI/CD (Continuous Integration / Continuous Deployment) in Salesforce?
A. It allows manual deployments via the Setup menu
B. It ensures automated testing and deployment, reducing errors
C. It increases governor limits for Apex code
D. It allows developers to write code without using version control

PDI User Interface

1. Visualforce

Visualforce is a legacy framework used primarily for Salesforce Classic and specific custom UI requirements. It utilizes <apex> tags to build pages and integrates with Apex Custom Controllers to manage dynamic data and business logic. While modern development has shifted toward Lightning, Visualforce remains a necessary skill for managing existing enterprise UI debt.

2. Lightning Web Components (LWC)

Lightning Web Components (LWC) represent the modern standard for Salesforce UI development, built on standard web technologies like HTML, JavaScript, and CSS. LWC offers superior performance due to its native browser support and lightweight footprint. It leverages the Lightning Data Service (LDS) for data access, which provides reactive data binding and built-in caching.

3. User Experience Optimization

Optimizing the user interface directly impacts user adoption and system efficiency. Techniques such as using Dynamic Forms allow for the conditional display of fields, which reduces page clutter. Furthermore, developers should reduce server calls by utilizing the @wire decorator for reactive data binding and focusing on efficient component design to ensure fast page load times.

4. Lightning Components (Aura Components)

Aura Components preceded LWC as the primary component framework. While LWC is preferred for all new projects due to its faster rendering and standard-based code, Aura is still present in legacy Lightning pages and Experience Cloud. LWC's streamlined single-module structure makes it more scalable and easier for modern developers to maintain than the Salesforce-specific Aura model.

5. Standard Controllers in Visualforce

Standard Controllers are a high-value tool for performing basic CRUD operations on objects without writing custom Apex code. They are the ideal choice for simple pages that only require standard record behavior. Developers must transition to Custom Controllers only when the UI requires complex logic or queries that exceed standard object capabilities.

6. Lightning Record Pages & App Builder

The Lightning App Builder is a declarative tool used to customize the user experience via a drag-and-drop interface. It allows for the placement of both standard and custom components, including LWC and Aura, onto record pages. This flexibility enables architects to tailor the interface to specific user needs without the overhead of custom code.

7. LWC Data Access: @wire and Apex Calls

7.1. LWC data retrieval is managed through three distinct methods. Developers should prioritize Lightning Data Service via components like `lightning-record-form` and `lightning-record-view-form` for standard record operations without Apex. For more reactive data binding, the @wire service fetches data from Apex methods and automatically updates the UI when the underlying data changes.

7.2. Imperative Apex Calls are reserved for user-triggered actions, such as a button click, providing manual control over when a query executes. @wire remains the preferred approach for most data-fetching scenarios, ensuring a dynamic and responsive interface. Together, these tools form a balanced ecosystem of declarative and programmatic options that define the modern Salesforce development landscape.

8. User Interface Practice Question

Q1: What is the correct way to reference a Standard Controller in Visualforce?
A. `<apex:page controller="AccountController">`
B. `<apex:page standardController="Account">`
C. `<apex:page recordSetController="Account">`
D. `<apex:page standardController="AccountController">`

Q2: What is the purpose of a Custom Controller in Visualforce?
A. To allow Visualforce pages to interact with Apex classes
B. To use built-in Salesforce functionality without writing Apex
C. To enforce Salesforce sharing rules automatically
D. To automatically generate user interfaces

Q3: Which Visualforce tag is used to create an input field that binds to an Apex variable?
A. `<apex:outputText>`
B. `<apex:commandButton>`
C. `<apex:inputText>`
D. `<apex:detail>`

Q4: Which of the following is NOT a valid benefit of Lightning Web Components (LWC)?
A. Uses modern web standards
B. Requires Aura framework to function
C. Provides better performance than Visualforce
D. Can be used in Lightning App Builder

Q5: What is the correct way to handle a button click event in LWC?
A. Using `@track`
B. Using `@wire`
C. Using `onclick={handleClick}`
D. Using `<apex:commandButton>`

Q6: Which decorator should be used in LWC to fetch Salesforce data using Apex?
A. `@track`
B. `@api`
C. `@wire`
D. `@future`

Q7: What is the primary use of Lightning App Builder in Salesforce?
A. To create and customize Lightning pages with drag-and-drop functionality
B. To build Apex classes for handling backend logic
C. To define Visualforce pages
D. To deploy metadata to production

Q8: Which of the following is a best practice for optimizing Lightning page performance?
A. Loading all data at once
B. Using Lightning Data Service (LDS)
C. Running SOQL queries inside a for-loop
D. Using multiple triggers per object

Learning Path & Study Advice

A recommended preparation path begins with developing a clear understanding of Salesforce platform architecture and the metadata-driven approach used to build applications. Learners should first become comfortable with the core data model, object relationships, and the ways in which configuration features shape application behavior.

After establishing this foundation, candidates should study how business logic and automation are implemented within the platform. Understanding how declarative automation tools operate provides an important conceptual basis before moving into programmatic development techniques.

Next, learners should focus on user interface customization and the ways applications present data to users. Understanding the interaction between data structures, automation, and the interface layer helps reinforce how complete applications function within the platform.

Finally, preparation should emphasize testing and debugging practices. Developing the habit of validating logic, identifying errors, and maintaining reliable code helps reinforce long-term development skills and ensures that solutions remain stable as applications evolve.

Who This PDF Is For

This document is intended for individuals preparing to develop applications on the Salesforce platform and seeking a structured overview of the knowledge areas associated with the Platform Developer I certification.

It is suitable for aspiring Salesforce developers, junior platform developers, and administrators expanding into programmatic development. Professionals with a basic understanding of programming concepts, cloud platforms, or Salesforce configuration will benefit most from this material. The document also serves as a foundational reference for those beginning a technical learning path within the Salesforce ecosystem.

Call To Action

This document provides an overview of structured learning and certification preparation approaches. For learners seeking clear knowledge organization, guided study planning, and exam-focused practice resources, AAAdemy offers a comprehensive platform to support independent and effective learning.

Explore additional training materials, study guidance, and practice resources at:

https://www.aaademy.com/Platform-Developer-I/PDI.html

Online Flashcards (Quizlet):

https://quizlet.com/user/AAAdemy/folders/pdi-platform-developer-i-exam-flashcards-aaademy?i=6zfa5t&x=1xqt

Attachment: Answers by Knowledge Point

Developer Fundamentals Practice Question
A1: Answer: D. Custom Objects can have their own fields, relationships, and layouts.
Explanation:

- Standard Objects (e.g., Account, Contact) are predefined by Salesforce and can be customized but not deleted.
- Custom Objects are created by users and can include custom fields, relationships, layouts, and more.
- Both Standard and Custom Objects can have Master-Detail relationships.

A2: Answer: B. Master-Detail Relationship allows sharing settings to be inherited, while Lookup does not.
Explanation:

- In a Master-Detail Relationship, the child record inherits the owner and sharing rules of the parent record.
- If the parent is deleted, the child records are also deleted (cascading delete).
- Lookup Relationships do not enforce sharing rules or ownership inheritance and allow null values.
- Roll-up Summary Fields are only available in Master-Detail relationships.

A3: Answer: C. Formula
Explanation:

- A Formula Field automatically updates based on other fields' values.

- Unlike Number fields, formulas are read-only and cannot be edited manually.

A4: Answer: A. `SELECT Name FROM Account WHERE Industry = 'Technology'`
Explanation:

- SOQL (Salesforce Object Query Language) uses `SELECT ... FROM ... WHERE ...` syntax.
- Double equals (`==`) is incorrect in SOQL; use `=` instead.
- `FIND ... RETURNING` is SOSL syntax, not SOQL.
- `LIKE '%Technology%'` works in SOQL but is used for partial matches, which is unnecessary here.

A5: Answer: B. `Trigger.New`
Explanation:

- `Trigger.New` contains the new values of records in before insert, after insert, before update, and after update triggers.
- `Trigger.Old` contains the old values of records before an update or delete.
- `Trigger.NewMap` and `Trigger.OldMap` are only available in update and delete triggers.

A6: Answer: B. To prevent a single tenant from consuming excessive shared resources.
Explanation:

- Governor Limits exist because Salesforce is a multi-tenant platform.
- These limits ensure that no single org can overuse resources, impacting others.
- Example: SOQL query limit = 100 per transaction.

A7: Answer: D. Trigger Apex
Explanation:

- Triggers are synchronous; they execute immediately when a record changes.
- Future Methods, Queueable Apex, and Batch Apex are all asynchronous processing techniques in Salesforce.

A8: Answer: B. Using Change Sets
Explanation:

- Change Sets are the recommended method to deploy metadata changes (Apex, Objects, Fields, etc.) between environments.
- Data Loader is used for importing/exporting data, not code.
- Production code cannot be edited directly.

A9: Answer: C. `with sharing`
Explanation:

- `with sharing` enforces Salesforce's sharing rules, ensuring that users can only access records they have permission to view.
- `without sharing` means the class ignores sharing rules.
- `private sharing` and `global sharing` are invalid keywords.

A10: Answer: D. `Test.loadData()`
Explanation:

- `Test.loadData()` loads test data from a static resource, allowing bulk testing without affecting live records.
- `Test.startTest()` is used for isolating test execution but does not create records.
- `@isTest` is an annotation used for defining test classes.
- `System.insert` directly inserts records, which can impact test isolation.

Process Automation and Logic Practice Question
A1: Answer: C. Flow Builder
Explanation:

- Flow allows for complex, multi-step logic including user inputs, decision-making, and loops.
- Process Builder is more limited and does not support user interaction.
- Workflow Rules are simple automation tools for field updates and emails.

- Approval Processes are used for record approvals.

A2: Answer: B. Creating a new record
Explanation:

- Workflow Rules can only perform field updates, email alerts, task assignments, and outbound messages.
- Creating new records requires Process Builder or Flow.

A3: Answer: B. Flow allows user input and screens
Explanation:

- Flow supports screen interactions and allows users to input data, which Process Builder does not.
- Process Builder is being replaced by Flow because Flow can handle more complex logic.
- Before-save record updates are handled in Flow, not Process Builder.

A4: Answer: C. Implementing a multi-step manager approval workflow
Explanation:

- Approval Processes are designed for multi-step approval workflows, requiring manual approval from managers.
- Flows and Process Builder handle automatic updates and related records.

A5: Answer: B.

```
trigger ContactTrigger on Contact (before insert) {
  for (Contact con : Trigger.new) {
     con.Description = 'Welcome!';
  }
}
```

Explanation:

- Before Insert triggers allow modifying records before they are saved to the database.
- After Insert triggers cannot modify records directly.
- Before Update runs for existing records, not new ones.

A6: Answer: A. To avoid recursive execution of triggers
Explanation:

- Trigger Frameworks help prevent triggers from running multiple times during bulk operations.
- Salesforce recommends only one trigger per object, which should delegate logic to a handler class.

A7: Answer: C. Batch Apex
Explanation:

- Batch Apex is optimized for handling large data sets in chunks (up to 200 records per batch).
- Future Methods cannot process records in batches.
- Queueable Apex can handle complex objects but is not optimized for bulk data processing.

A8: Answer: B. Queueable Apex allows chaining of jobs
Explanation:

- Queueable Apex supports chaining multiple jobs, whereas Future Methods cannot.
- Future Methods are simpler but less flexible than Queueable Apex.

A9: Answer: A.

System.schedule('Daily Job', '0 0 12 * * ?', new ScheduledJob());

Explanation:

- `System.schedule()` is the correct method to schedule Apex jobs in Salesforce.
- The cron expression '0 0 12 * * ?' means every day at 12 PM.
- `System.scheduleBatch()` is for Batch Apex, not Scheduled Jobs.

User Interface Practice Question
A1: Answer: B. `<apex:page standardController="Account">`
Explanation:

- Standard Controllers allow Visualforce pages to work with standard Salesforce objects without custom Apex code.

- The correct syntax is:

```
<apex:page standardController="Account">
  <h1>{!Account.Name}</h1>
</apex:page>
```

- Option A and D are incorrect because they reference a custom controller, not a standard controller.
- Option C is used for list views (`recordSetController`).

A2: Answer: A. To allow Visualforce pages to interact with Apex classes
Explanation:

- Custom Controllers are Apex classes that define the logic behind a Visualforce page.
- They are used when more control over the logic is needed (e.g., querying records, modifying data).
- Standard Controllers (Option B) provide built-in Salesforce logic.
- Sharing rules (Option C) are not enforced unless `with sharing` is specified.

A3: Answer: C. `<apex:inputText>`
Explanation:

- `<apex:inputText>` binds a text field to an Apex variable.
- Example:

```
<apex:inputText value="{!name}"/>
```

- Option A (`<apex:outputText>`) is used for displaying text.
- Option B (`<apex:commandButton>`) is used for submitting actions.
- Option D (`<apex:detail>`) is used to display record details.

A4: Answer: B. Requires Aura framework to function
Explanation:

- LWC does NOT require Aura; it is built natively on Web Standards (HTML, JavaScript, CSS).
- LWC advantages include:

- Better performance (C) compared to Visualforce.
- Drag-and-drop support (D) in Lightning App Builder.
- Modern Web Standards (A).

A5: Answer: C. Using `onclick={handleClick}`
Explanation:

- LWC uses standard JavaScript event handling.
- Example:

```
<lightning-button label="Click Me" onclick={handleClick}></lightning-button>
```

- `@track` (A) is used for reactive properties (outdated).
- `@wire` (B) is for data fetching, not event handling.
- `<apex:commandButton>` (D) is for Visualforce.

A6: Answer: C. `@wire`
Explanation:

- `@wire` is used for calling Apex methods and Salesforce data.
- Example:

```
import { LightningElement, wire } from 'lwc';
import getAccounts from '@salesforce/apex/AccountController.getAccounts';

export default class AccountList extends LightningElement {
    @wire(getAccounts) accounts;
}
```

- `@track` (A) was used for reactive properties (deprecated).
- `@api` (B) is for public component properties.
- `@future` (D) is for asynchronous Apex execution, not LWC.

A7: Answer: A. To create and customize Lightning pages with drag-and-drop functionality
Explanation:

- Lightning App Builder allows developers/admins to drag and drop Lightning Components into a page.
- It is not used for writing Apex (`B`), defining Visualforce (`C`), or deployment (`D`).

A8: Answer: B. Using Lightning Data Service (LDS)
Explanation:

- LDS optimizes data access by reducing unnecessary server calls.
- Loading all data at once (`A`) decreases performance.
- SOQL inside loops (`C`) is a bad practice (causes governor limits).
- Multiple triggers (`D`) can cause recursion issues.

Testing, Debugging, and Deployment Practice Question
A1: Answer: B. 75%
Explanation:

- Salesforce requires at least 75% of all Apex code to be covered by unit tests before deployment to production.
- However, 75% is the minimum requirement; aiming for 100% coverage is recommended.
- Test methods should also include positive, negative, and bulk data tests.

A2: Answer: B. To execute test data creation once for all test methods in a class
Explanation:

- `@TestSetup` methods create test data once and reuse it across multiple test methods, improving test efficiency.
- Example:

```
 @TestSetup
static void setupTestData() {
    Account acc = new Account(Name = 'Test Account');
    insert acc;
}
```

- This avoids repeating data creation inside every test method, making tests faster and more maintainable.

A3: Answer: B. Use `Test.startTest()` and `Test.stopTest()`
Explanation:

- `Test.startTest()` and `Test.stopTest()` create a separate execution context, allowing asynchronous code (e.g., Future, Queueable Apex) to run during testing.

- Example:

```
@isTest
public class AsyncTest {
    @isTest
    static void testFutureMethod() {
        Test.startTest();
        FutureClass.someFutureMethod();
        Test.stopTest();
        System.assertEquals(1, [SELECT COUNT() FROM Async_Result__c]);
    }
}
```

- Without `Test.stopTest()`, the Future method may not execute during testing.

A4: Answer: A. Open the Debug Logs in Developer Console
Explanation:

- System.debug() messages appear in Debug Logs when logging is enabled in Developer Console.

- To enable logs:

    - Open Developer Console → Debug > View Logs.

    - Set Logging Level to Debug for a specific user.

- Option B & C are incorrect as debug logs are not stored as records.

A5: Answer: B. Use "Debug Only" filter in the Logs tab
Explanation:

- The "Debug Only" checkbox in Developer Console filters out non-debug log entries, making it easier to analyze logs.

- Option A (SOQL Analyzer) is for optimizing SOQL queries.

- Option C (FATAL Log Level) would filter only fatal errors, not debug messages.

A6: Answer: A. Change Sets
Explanation:

- Change Sets allow manual selection of metadata and deployment between Salesforce environments via Setup.
- SFDX CLI (`B`) is a command-line tool, not a UI-based metadata selection tool.
- Workbench (`C`) is used for querying and making API calls, not deployments.

A7: Answer: B. They allow deployment between unrelated Salesforce orgs
Explanation:

- Change Sets can only be used between connected sandbox and production orgs (same Salesforce environment).
- Option C (Validation Required) is correct because Change Sets must be validated before deployment.
- Option D is correct since not all metadata types (e.g., Profiles, Sharing Rules) are supported.

A8: Answer: B. `sfdx force:source:push`
Explanation:

- This command pushes metadata from a local project into a Salesforce org.
- Option A (`bulk:upsert`) is for data operations, not metadata deployment.
- Option C (`auth:web:login`) is used to log in to Salesforce.
- Option D (`org:open`) opens an org in the browser.

A9: Answer: B. It ensures automated testing and deployment, reducing errors
Explanation:

- CI/CD automates the testing and deployment process, reducing human errors and ensuring high code quality.
- Option A is incorrect because CI/CD automates, not requires manual setup.

- Option C is incorrect as Governor Limits are fixed and do not change with CI/CD.

- Option D is incorrect because CI/CD heavily relies on version control (e.g., Git).