🤖 How to use AI

Creating Proper and Valuable Prompts

A Practical Workshop for QA Automation Engineers

Duration: 1 hour

📋 Workshop Agenda

1. **Introduction to AI Prompting** (5 min)
2. **The CLEAR Framework** (10 min)
3. **Common Prompting Mistakes** (8 min)
4. **Advanced Prompting Techniques** (12 min)
5. **QA-Specific Use Cases & Examples** (15 min)
6. **Hands-on Practice & Tips** (8 min)
7. **Q&A** (2 min)

🎯 Why Proper Prompting Matters

❌ Poor Prompts Lead To:

- Vague or irrelevant responses
- Wasted time iterating
- Incomplete solutions
- Frustration and low confidence
- Missed opportunities

✅ Good Prompts Result In:

- Precise, actionable answers
- Time saved (3-5x faster)
- Complete, working solutions
- Increased productivity
- Better code quality

💡 **Key Insight:** The quality of AI output is directly proportional to the quality of your input prompt.

🎨 The CLEAR Framework for Effective Prompts

Letter	Component	Description
C	**Context**	Provide background information about your situation, role, and environment
L	**Level**	Specify your expertise level and desired complexity of response
E	**Examples**	Include examples of what you want or reference formats
A	**Action**	Clearly state what you want the AI to do
R	**Requirements**	List specific constraints, preferences, or output format

💡 **Pro Tip:** You don't need all 5 elements every time, but including more elements typically yields better results!

🎨 CLEAR Framework in Action

❌ Vague Prompt:

"Help me with Playwright tests"

✅ CLEAR Prompt:

Context: I'm a QA engineer working on an e-commerce web app using Playwright with TypeScript.

Level: Intermediate level - familiar with basic Playwright but new to advanced patterns.

Examples: I've seen Page Object Model patterns but haven't implemented them yet.

Action: Create a reusable Page Object Model for a login page.

Requirements: Include error handling, waiting strategies, and TypeScript types. Follow best practices for maintainability.

Result: The second prompt will give you a complete, production-ready solution instead of generic advice!

⚠️ Common Prompting Mistakes

1. Being Too Vague

❌ **Bad:** "Write a test"

✅ **Good:** "Write a Playwright test in TypeScript that validates login functionality with valid/invalid credentials and checks for proper error messages"

2. No Context About Tech Stack

❌ **Bad:** "How do I wait for elements?"

✅ **Good:** "In Playwright with TypeScript, what's the best practice for waiting for dynamic elements loaded via AJAX? Show me examples using locator strategies."

3. Assuming AI Knows Your Codebase

❌ **Bad:** "Fix the test"

✅ **Good:** "Here's my failing test: [paste code]. It fails with error: [paste error]. The test is for a checkout flow. How can I fix it?"

⚠️ Common Mistakes (Continued)

4. Not Specifying Output Format

❌ **Bad:** "Explain API testing"

✅ **Good:** "Create a step-by-step guide for API testing with Playwright. Include code examples, explain assertions, and provide a complete working test. Format as: setup, test cases, assertions."

5. Asking Multiple Things at Once

❌ **Bad:** "Help me with Playwright setup, write tests, integrate CI/CD, and fix my bugs"

✅ **Good:** Break into separate prompts:

- First: "Help me set up Playwright with TypeScript..."
- Then: "Now write a test for..."
- Finally: "Integrate these tests into GitHub Actions..."

🚀 Advanced Technique #1: Role Playing

Concept: Tell the AI to act as a specific expert or persona to get specialized responses.

✅ Example:

```
"Act as a senior QA automation architect with 10 years of experience in
Playwright. I'm implementing a testing framework from scratch for a
microservices architecture. What are the must-have design patterns and folder
structure you'd recommend? Include rationale for each decision."
```

💡 **Why It Works:** Role-playing primes the AI to respond with expertise, best practices, and nuanced insights that match the specified persona.

Useful Personas for QA:

- "Act as a code reviewer focused on test quality..."
- "Act as a DevOps engineer specializing in CI/CD pipelines..."
- "Act as a security testing expert..."
- "Act as a performance testing consultant..."

🚀 Advanced Technique #2: Chain-of-Thought

Concept: Ask the AI to explain its reasoning step-by-step, leading to more thorough and accurate solutions.

✅ Example:

```
"I need to test a complex multi-step checkout flow in Playwright. Walk me
through your thought process step-by-step: 1. How would you structure the test?
2. What challenges might arise? 3. How would you handle each challenge? 4. Then
provide the final implementation with explanations."
```

Key Phrases to Trigger Chain-of-Thought:

- "Explain your reasoning step-by-step..."
- "Walk me through the thought process..."
- "Let's think through this systematically..."
- "First analyze the problem, then provide the solution..."
- "What are the trade-offs of different approaches?"

💡 **Benefit:** You get not just code, but understanding of WHY certain decisions were made.

🚀 Advanced Technique #3: Iterative Refinement

Concept: Start broad, then progressively refine with follow-up prompts.

🔄 The Refinement Pattern:

1️⃣ **Initial Prompt:**

```
"Create a Playwright test for user registration"
```

2️⃣ **Refinement 1:**

```
"Add validation for email format, password strength, and duplicate users"
```

3️⃣ **Refinement 2:**

```
"Extract the validation logic into reusable helper functions"
```

4️⃣ **Refinement 3:**

```
"Add TypeScript types and JSDoc comments for better maintainability"
```

💡 **Pro Tip:** Don't try to get perfection in one prompt. Iterate and refine!

🚀 Advanced Technique #4: Adding Constraints

Concept: Explicitly state what you DON'T want, limitations, and boundaries.

✅ Example with Constraints:

```
"Create a Playwright test data generator for user profiles. REQUIREMENTS: - Use
faker.js for realistic data - Generate 50 unique users - Include: name, email,
phone, address CONSTRAINTS: - Do NOT use deprecated faker methods - Do NOT
hardcode any values - Must be TypeScript with proper types - Email domains
should be realistic (not @example.com) - Phone numbers must match US format
OUTPUT FORMAT: - Export as JSON array - Include a seed function for
reproducibility"
```

💡 **Why It Works:** Constraints prevent common mistakes and ensure the solution fits your exact needs.

🧪 QA Use Case #1: Test Generation

Scenario: You need to create tests for a new feature

✅ Effective Prompt:

```
"I'm testing a new shopping cart feature in our e-commerce app using Playwright
+ TypeScript. FEATURE DETAILS: - Users can add/remove items - Cart shows total
price with tax - Apply discount codes - Quantity limits (max 10 per item)
GENERATE: 1. A comprehensive test suite covering happy path and edge cases 2.
Use Page Object Model pattern 3. Include data-driven tests for multiple
discount codes 4. Add comments explaining each test's purpose TECH STACK: -
```

```
Playwright 1.40+ - TypeScript - Jest assertions Please organize tests by
scenario and include setup/teardown."
```

Result: You'll get a complete, organized test suite ready to use!

🐛 QA Use Case #2: Debugging Flaky Tests

Scenario: Your test is intermittently failing

✅ Effective Prompt:

```
"My Playwright test is flaky - it fails ~30% of the time. TEST CODE: [paste
your test code here] ERROR MESSAGE: TimeoutError: Waiting for selector
'#submit-button' to be visible CONTEXT: - Testing a React SPA with dynamic
content - Button appears after API call completes - Happens more on slow
networks - Using playwright.config with 30s timeout ANALYZE: 1. Identify likely
root causes of the flakiness 2. Suggest specific fixes with code examples 3.
```

```
Recommend best practices to prevent similar issues 4. Show how to add better
waiting strategies Environment: Playwright 1.40, TypeScript, running in CI/CD"
```

💡 **Key Elements:** Include code, error, context, and specific request for analysis.

👀 QA Use Case #3: Code Review & Optimization

✅ Effective Prompt:

```
"Review this Playwright test code as a senior QA engineer. CODE: [paste your
test code] REVIEW FOR: 1. Best practices adherence 2. Potential reliability
issues 3. Performance optimization opportunities 4. Maintainability and
readability 5. Security concerns in test data PROVIDE: - Specific line-by-line
feedback - Code examples for improvements - Severity rating (Critical/Major/
Minor) - Refactored version of the code CONTEXT: - Team of 5 QAs, various skill
levels - Tests run in CI/CD on every PR - Test suite currently takes 45 minutes
- We value maintainability over brevity"
```

💡 **Pro Tip:** Use AI as a 24/7 code reviewer before asking colleagues!

📚 QA Use Case #4: Documentation & Comments

✅ Effective Prompt:

```
"Generate comprehensive documentation for this test suite. CODE: [paste your
test files] GENERATE: 1. README.md with: - Overview of test coverage - Setup
instructions - How to run tests (locally & CI) - Folder structure explanation -
Troubleshooting guide 2. Inline JSDoc comments for: - All test functions -
Helper functions - Page Objects 3. A test plan document mapping: - Features to
test files - Coverage gaps - Priority levels AUDIENCE: New team members with
basic Playwright knowledge TONE: Professional but friendly FORMAT: Markdown
with code examples"
```

Result: Complete documentation that would take hours to write manually!

🌐 QA Use Case #5: API Test Automation

✅ Effective Prompt:

```
"Create API tests for a RESTful user management service using Playwright's API
testing. API ENDPOINTS: - POST /api/users (create user) - GET /api/users/:id
(get user) - PUT /api/users/:id (update user) - DELETE /api/users/:id (delete
user) AUTHENTICATION: Bearer token (provided in env vars) TEST REQUIREMENTS: 1.
Test all CRUD operations 2. Validate response schemas 3. Test error scenarios
(400, 401, 404) 4. Test data validation rules 5. Performance assertions
(response time < 200ms) INCLUDE: - Request/response logging - Reusable API
client class - TypeScript interfaces for payloads - Test data factories -
Cleanup after tests Follow AAA pattern (Arrange-Act-Assert) with clear test
names."
```

⚙️ QA Use Case #6: CI/CD Pipeline Setup

✅ Effective Prompt:

```
"Create a GitHub Actions workflow for Playwright tests. REQUIREMENTS: - Run on:
push to main, all PRs - Test on: Ubuntu, multiple browsers (chromium, firefox)
- Node version: 18.x - Install dependencies and Playwright browsers - Run tests
in parallel - Upload test reports and screenshots as artifacts - Comment test
results on PRs - Fail fast if critical tests fail - Cache dependencies for
faster runs CURRENT SETUP: - Package manager: npm - Test command: npm run
test:e2e - Tests in: tests/ directory - Config: playwright.config.ts ALSO
```

```
INCLUDE: - Environment variables handling - Secrets management for API keys -
Conditional running (only on changed files) - Slack notification on failures"
```

📝 Reusable Prompt Templates

Template 1: Code Generation

```
"Create [WHAT] for [FEATURE] using [TECH STACK]. REQUIREMENTS: - [Requirement
1] - [Requirement 2] - [Requirement 3] CONSTRAINTS: - [Constraint 1] -
[Constraint 2] INCLUDE: - [Element 1] - [Element 2] Context: [Your specific
situation]"
```

Template 2: Debugging

```
"Debug this [TYPE] issue: CODE: [paste code] ERROR: [paste error] EXPECTED:
[what should happen] ACTUAL: [what happens instead] CONTEXT: - Environment:
[details] - Frequency: [always/sometimes] - Recent changes: [any changes]
Please explain the root cause and provide a fix."
```

📝 More Reusable Templates

Template 3: Refactoring

```
"Refactor this code for [GOAL]: CURRENT CODE: [paste code] IMPROVE: - [Aspect 1
- e.g., readability] - [Aspect 2 - e.g., performance] - [Aspect 3 - e.g.,
maintainability] CONSTRAINTS: - Keep existing functionality - Maintain backward
compatibility - [Other constraints] Explain each change and why it's better."
```

Template 4: Learning

```
"Explain [CONCEPT] in the context of [YOUR DOMAIN]. MY LEVEL: [Beginner/
Intermediate/Advanced] INCLUDE: - Simple explanation with analogies - Real-
world example from [YOUR DOMAIN] - Code example showing usage - Common pitfalls
to avoid - Resources for deeper learning Make it practical and actionable."
```

✨ Best Practices Summary

Do's ✅	Don'ts ❌
Be specific and detailed	Don't be vague or ambiguous
Provide context and constraints	Don't assume AI knows your setup
Include code/error examples	Don't just describe problems
Specify desired format	Don't expect mind-reading
Break complex tasks into steps	Don't ask multiple things at once
Iterate and refine	Don't expect perfection first try
Use role-playing	Don't use generic prompts
Request explanations	Don't just copy code blindly

🎯 **Golden Rule:** Treat AI like a skilled colleague who needs clear requirements to help you effectively!

💪 Hands-on Practice Exercise

🎯 Your Challenge (5 minutes)

Transform this poor prompt into an excellent one using the CLEAR framework:

Poor Prompt:

"Help me test a form"

📝 Think About:

- What context is missing?
- What technical details should you add?

- What specific action do you want?
- What requirements or constraints apply?
- What examples would help?

💡 **Tip:** Try to apply at least 4 out of 5 CLEAR elements!

✅ Exercise Solution Example

Improved Prompt Using CLEAR:

```
"I'm a QA engineer testing a multi-step registration form in a React app using
Playwright with TypeScript. FORM DETAILS (Context): - 3 steps: Personal info →
Account details → Confirmation - Fields: name, email, password, phone, address
- Validation on each step before proceeding - Form data persists if user goes
back MY LEVEL: Intermediate - comfortable with Playwright basics ACTION NEEDED:
Create a comprehensive test suite for this form REQUIREMENTS: 1. Test happy
path (complete registration) 2. Test field validation (required fields,
formats) 3. Test navigation between steps 4. Test data persistence 5. Test
error messages display 6. Use Page Object Model pattern 7. Include TypeScript
types 8. Add data-driven tests for different user scenarios EXAMPLES of
validation rules: - Email: must be valid format - Password: min 8 chars, 1
```

```
uppercase, 1 number - Phone: US format only Please structure with clear test
descriptions and comments."
```

⚡ Quick Wins - Start Today!

🚀 Immediate Actions:

1. **Save Templates**
   Create a "prompts.md" file with your go-to templates
2. **Add Context Header**
   Start every prompt with your tech stack and role
3. **Use "Act as..."**
   Begin prompts with role definitions
4. **Iterate More**
   Don't settle for first response - refine!
5. **Share with Team**
   Create a team prompt library

📊 Measure Success:

- **Time Saved**
  Track how much faster you complete tasks
- **Code Quality**
  Compare AI-assisted vs manual code
- **Learning Curve**
  Note new concepts learned via AI
- **Iterations**
  Aim to reduce prompt iterations over time
- **Team Adoption**
  Track usage across team

🎯 **Goal:** Aim to reduce task completion time by 40-50% within first month!

⚠️ Common Pitfalls to Avoid

1. 🤖 Over-Reliance Without Understanding

- **Problem:** Blindly copying AI code without understanding
- **Solution:** Always ask "Explain why this works" after getting code

2. 🔒 Security & Privacy Issues

- **Problem:** Sharing sensitive data, credentials, or proprietary code
- **Solution:** Sanitize data, use placeholders, check company policies

3. 📅 Outdated Information

- **Problem:** AI may suggest deprecated methods or old versions
- **Solution:** Specify versions, verify against official docs

4. 🎯 Not Verifying Output

- **Problem:** AI can make mistakes or misunderstand requirements
- **Solution:** Always test, review, and validate generated code

5. 🚫 Using AI as Search Engine

- **Problem:** Asking AI to find documentation instead of crafting solutions
- **Solution:** Use AI for creation, analysis, and problem-solving

📚 Additional Resources

🔗 Learning Resources:

- **Prompt Engineering Guide:** promptingguide.ai
- **OpenAI Cookbook:** github.com/openai/openai-cookbook
- **Learn Prompting:** learnprompting.org
- **AI Testing Tools:** Testim.io AI features, Applitools

🛠️ Tools to Explore:

- **ChatGPT:** General purpose, great for code and explanations
- **GitHub Copilot:** In-IDE code suggestions
- **Cursor AI:** AI-powered code editor

- **Claude:** Excellent for analysis and detailed explanations
- **Tabnine:** AI code completion

👥 Community:

- Join QA automation communities on Discord/Slack
- Share prompts and learnings with your team
- Create an internal wiki of effective prompts

🎯 Key Takeaways

1️⃣ Master the CLEAR Framework

Context + Level + Examples + Action + Requirements = Better Results

2️⃣ Be Specific, Not Generic

The more context and detail you provide, the better the output

3️⃣ Iterate and Refine

First response is rarely perfect - keep improving

4️⃣ Verify and Understand

Never blindly trust - always validate and learn

5️⃣ Build Your Prompt Library

Save and share effective prompts with your team

❓ Questions & Answers

Let's discuss your specific use cases!

📧 Contact: [Your Email]

💼 LinkedIn: [Your Profile]

🐙 GitHub: [Your Repo]

🙏 Thank You!

Start Prompting Better Today!

Remember: The quality of your AI interactions
depends on the quality of your prompts.

💪 Practice → 🎯 Improve → 🚀 Excel