ArcaneQuest Language Reference

🎯 Language Syntax

Keywords & Their Meanings

ArcaneQuest Keyword	Traditional Equivalent	Purpose
summon	import	Import modules
quest	def	Define function
reward	return	Return value
attack	print	Output/print
scout	input	Get user input
spot	if	Conditional if
counter	elif	Else-if
dodge	else	Else clause
replay	while	While loop
farm	for	For loop
guild	class	Define class
case	case	Match case
embark	try	Try block
gameOver	except	Exception handler
savePoint	finally	Finally block
skipEncounter	continue	Continue loop
escapeDungeon	break	Break loop

Data Types

- potion - int
- elixir - float
- fate - boolean
- scroll - string

Operators

- Arithmetic: +, -, *, /
- Comparison: <, >, <=, >=, ==, !=
- Logical: and, or, not
- Assignment: =, +=, -=, *=, /=

Comments

Use --> for single-line comments:

--> This is a comment

📝 Code Examples

Hello World

```
attack("Hello, World!")
```

Import Modules

```
summon random, sys
```

Function Definition

```
quest greet(name):
    attack("Hello,", name)
    reward "Welcome!"
```

Variables and Input

```
name = scout("Enter your name: ")
attack("Hello,", name)
```

Conditional Statements

```
spot (health > 50):
    attack("You are healthy!")
counter (health > 20):
    attack("You are wounded!")
dodge:
    attack("Critical condition!")
```

Loops

--> While loop

```
replay (count < 10):
    attack(count)
    count += 1
```

--> For loop

```
farm item in inventory:
    attack("Found:", item)
```

Classes

```
guild Hero:
    quest __init__(name):
        attack("Hero created:", name)
```

Exception Handling

```
embark:
    risky_operation()
gameOver ValueError:
    attack("Invalid value!")
gameOver:
    attack("Unknown error!")
savePoint:
    attack("Cleanup complete")
```

Function Calls

```
scroll("message")
sys.exit(0)
player.take_damage(10)
```

🔧 IDE Usage

Interface Components

1. **Source Editor** (Left Panel)
   - Write your ArcaneQuest code here
   - Supports .aq file extensions
2. **Scanner Output** (Right Top)
   - Shows tokenized output
   - Displays token types and line numbers

3. **Parser Output** (Right Bottom)
    - Shows the Abstract Syntax Tree (AST)
    - Displays parsing errors if any

Buttons

- **Load** - Open an .aq file
- **Scan** - Tokenize the source code
- **Parse** - Parse and validate syntax
- **Clear** - Clear all panels

⚠ Syntax Rules

Indentation

- **Consistent indentation is required**
- First indent sets the standard (e.g., 4 spaces)
- All subsequent indents must match exactly

🐛 Error Messages

The parser provides detailed error messages:

- **Line numbers** for easy debugging
- **Clear descriptions** of what went wrong
- **Partial parse tree** even when errors occur

Example error output:

⚠ Parsing failed: 2 error(s)

Line 3: Expected ',' after module name

Line 5: Invalid statement: bare identifier 'test' cannot stand alone

🔮 Advanced Features

Operator Precedence

1. or (lowest)
2. and
3. not
4. ==, !=, <, >, <=, >=
5. +, -
6. *, / (highest)

Attribute Access

player.health

sys.exit

math.sqrt(16)

Nested Structures

```
quest complex_function(x, y):
    spot (x > 0):
        replay (y < 10):
            attack(x, y)
            y += 1
    reward x + y
```