ATLASENGINE COMPLETE GUIDE - PART 2

T# Language: Complete Command Reference

All 334 Commands Documented

PART 3: T# LANGUAGE - COMPLETE COMMAND REFERENCE

COMMAND CATEGORIES:

1. **Text Output** (10 commands)
2. **Variables & Assignment** (15 commands)
3. **Math Operations** (30 commands)
4. **Control Flow** (15 commands)
5. **String Operations** (20 commands)
6. **List Operations** (25 commands)
7. **Graphics 2D** (20 commands)
8. **Graphics 3D** (80 commands)
9. **Game Mechanics** (60 commands)
10. **Physics** (25 commands)
11. **AI & Behavior** (15 commands)
12. **Advanced** (19 commands)

Total: 334 Commands

CATEGORY 1: TEXT OUTPUT (10 COMMANDS)

1. say

Purpose: Display normal text **Syntax:** `say "text"` or `say variable` **Examples:**

```
say "Hello, World!"
say "Welcome to AtlasEngine"

name is "Player"
say name                    → Player
say "Your name: " name      → Your name: Player

score is 100
say "Score: " score         → Score: 100
```

Output: Normal text (white)

---

2. shout

Purpose: Display loud/emphasized text **Syntax:** `shout "text"` **Examples:**

```
shout "GAME OVER!"
shout "VICTORY!"
shout "LEVEL UP!"
```

Output: RED, BOLD text

---

3. whisper

Purpose: Display quiet/subtle text **Syntax:** `whisper "text"` **Examples:**

```
whisper "hint: check the chest"
whisper "secret passage nearby"
whisper "(narrator voice)"
```

Output: Gray, italic, small text

---

4. show

Purpose: Display variable value with label **Syntax:** `show variable` **Examples:**

```
health is 75
show health                 → health: 75

score is 1000
show score                  → score: 1000
```

Output: Blue text with variable name

---

5. input

Purpose: Get user input **Syntax:** `input "prompt" into variable` **Examples:**

```
input "Enter your name:" into player_name
say "Hello, " player_name "!"

input "Choose 1-3:" into choice
when choice equals 1 {
    say "You chose option 1!"
}

input "Your age:" into age
say "You are " age " years old"
```

Output: Dialog box for input **Returns:** String or number (auto-detected)

---

6. print

Purpose: Debug output (same as say) **Syntax:** `print "text"` **Examples:**

```
print "Debug: x = " x
print "Loop iteration " i
```

Output: Normal text

---

7. clear

Purpose: Clear output window **Syntax:** `clear` **Examples:**

```
say "Old text"
clear
say "New text"              → Only "New text" visible
```

Effect: Clears text output

---

8. cleargraphics

Purpose: Clear graphics canvas **Syntax:** `cleargraphics` **Examples:**

```
drawcircle at 100, 100 radius 50 color "#ff0000"
cleargraphics
drawrect at 200, 200 size 100, 100 color "#00ff00"
```

Effect: Clears 2D graphics, keeps text

---

9. warn

Purpose: Display warning message **Syntax:** `warn "message"` **Examples:**

```
warn "Low health!"
warn "Ammo running out"
warn "Enemy nearby"
```

Output: YELLOW warning text in logs

---

10. error

Purpose: Display error message **Syntax:** `error "message"` **Examples:**

```
error "Invalid input"
error "File not found"
error "Cannot divide by zero"
```

Output: RED error text in logs

CATEGORY 2: VARIABLES & ASSIGNMENT (15 COMMANDS)

1. Variable Assignment (is)

Purpose: Create or set variable **Syntax:** `variable is value` **Examples:**

```
# Numbers
x is 10
y is 3.14
score is 1000

# Strings
name is "Alice"
message is "Hello"

# Math expressions
sum is 10 + 20              → 30
```

```
product is 5 * 6           → 30
result is x + y            → 13.14

# Boolean (treated as 1/0)
active is 1                → true
disabled is 0              → false
```

---

2. make

Purpose: Create variable (alias for 'is') **Syntax:** `make variable value` **Examples:**

```
make player_health 100
make score 0
make level 1
```

---

3. set

Purpose: Set variable value (alias) **Syntax:** `set variable value` **Examples:**

```
set health 100
set ammo 30
set lives 3
```

---

4. create

Purpose: Create variable (alias) **Syntax:** `create variable value` **Examples:**

```
create gold 500
create exp 0
```

---

5. change

Purpose: Change variable value **Syntax:** `change variable to value` **Examples:**

```
health is 100
change health to 75

score is 500
change score to 1000
```

---

6. increase

Purpose: Increase variable by amount **Syntax:** `increase variable by amount` **Examples:**

```
score is 100
increase score by 50        → score = 150

gold is 1000
increase gold by 250        → gold = 1250

level is 1
increase level by 1         → level = 2
```

---

7. decrease

Purpose: Decrease variable by amount **Syntax:** `decrease variable by amount` **Examples:**

```
health is 100
decrease health by 25       → health = 75

ammo is 30
decrease ammo by 1          → ammo = 29

lives is 3
decrease lives by 1         → lives = 2
```

---

8. increment

Purpose: Add 1 to variable **Syntax:** `increment variable` **Examples:**

```
count is 0
increment count             → count = 1
increment count             → count = 2
increment count             → count = 3
```

---

9. decrement

Purpose: Subtract 1 from variable **Syntax:** `decrement variable` **Examples:**

```
lives is 3
decrement lives            → lives = 2
decrement lives            → lives = 1
decrement lives            → lives = 0
```

---

10. remember

Purpose: Store value (memorable name) **Syntax:** `remember variable is value` **Examples:**

```
remember best_score is 5000
remember player_name is "Hero"
remember last_checkpoint is 3
```

---

11. forget

Purpose: Delete variable **Syntax:** `forget variable` **Examples:**

```
temp is 100
forget temp                → temp no longer exists
```

---

12. recall

Purpose: Display variable value **Syntax:** `recall variable` **Examples:**

```
score is 1000
recall score               → Shows: score = 1000
```

---

13. exists

Purpose: Check if variable exists **Syntax:** `exists variable` **Examples:**

```
x is 10
result is exists x         → 1 (true)
result is exists y         → 0 (false)
```

---

14. typeof

Purpose: Get variable type **Syntax:** `typeof variable` **Examples:**

```
x is 10
type is typeof x           → "number"

name is "Alice"
type is typeof name        → "string"
```

---

15. copy

Purpose: Copy variable value **Syntax:** `copy source to destination` **Examples:**

```
original is 100
copy original to backup

health is 75
copy health to old_health
```

CATEGORY 3: MATH OPERATIONS (30 COMMANDS)

Basic Arithmetic

1. add / plus

Syntax: `variable add value` or `result is a + b`

```
x is 10
x add 5                    → x = 15

sum is 10 + 20             → 30
total is x + y + z         → sum of x, y, z
```

2. subtract / minus

Syntax: `variable subtract value` or `result is a - b`

```
x is 100
x subtract 25              → x = 75

diff is 50 - 20            → 30
```

3. multiply / times

Syntax: `variable multiply value` or `result is a * b`

```
x is 5
x multiply 3                → x = 15

product is 6 * 7            → 42
```

4. divide

Syntax: `variable divide value` or `result is a / b`

```
x is 100
x divide 4                  → x = 25

quotient is 50 / 5          → 10
```

5. modulo

Syntax: `result is a modulo b` **Purpose:** Get remainder after division

```
remainder is 17 modulo 5    → 2
check is 10 modulo 2        → 0 (even)
check is 11 modulo 2        → 1 (odd)
```

Advanced Math

6. power

Syntax: `power base exponent` or `result is base ^ exponent`

```
result is power 2 8         → 256 (2^8)
squared is 5 ^ 2            → 25
cubed is 3 ^ 3              → 27
```

7. root

Syntax: `root number` **Purpose:** Square root

```
result is root 16           → 4
result is root 25           → 5
result is root 2            → 1.414...
```

8. squared

Syntax: `squared number`

```
result is squared 5         → 25
result is squared 10        → 100
```

9. cubed

Syntax: `cubed number`

```
result is cubed 3           → 27
result is cubed 5           → 125
```

Rounding

10. round

Syntax: `round number`

```
result is round 3.7         → 4
result is round 3.2         → 3
```

11. floor

Syntax: `floor number` **Purpose:** Round down

```
result is floor 3.9         → 3
result is floor 5.1         → 5
```

12. ceil / roundup

Syntax: `ceil number` **Purpose:** Round up

```
result is ceil 3.1          → 4
result is ceil 5.9          → 6
```

13. rounddown

Syntax: `rounddown number`

```
result is rounddown 7.8     → 7
```

Comparison

14. min

Syntax: `min a b` **Purpose:** Get smaller value

```
result is min 10 20         → 10
result is min 5 3           → 3
```

15. max

Syntax: `max a b` **Purpose:** Get larger value

```
result is max 10 20         → 20
result is max 5 3           → 5
```

16. clamp

Syntax: `clamp value min max` **Purpose:** Constrain value between min and max

```
result is clamp 150 0 100   → 100 (capped at max)
result is clamp -10 0 100   → 0 (raised to min)
result is clamp 50 0 100    → 50 (within range)
```

Trigonometry

17. sin

Syntax: `sin angle` **Purpose:** Sine (angle in degrees)

```
result is sin 90            → 1
result is sin 30            → 0.5
```

18. cos

Syntax: `cos angle` **Purpose:** Cosine (angle in degrees)

```
result is cos 0             → 1
result is cos 90            → 0
```

19. tan

Syntax: `tan angle` **Purpose:** Tangent (angle in degrees)

```
result is tan 45            → 1
```

Other Math

20. absolute

Syntax: `absolute number` **Purpose:** Get absolute value

```
result is absolute -10      → 10
result is absolute 15       → 15
```

21. sign

Syntax: `sign number` **Purpose:** Get sign (-1, 0, or 1)

```
result is sign 10          → 1
result is sign -5          → -1
result is sign 0           → 0
```

22. percent

Syntax: `percent value total` **Purpose:** Calculate percentage

```
result is percent 25 100    → 25
result is percent 1 4       → 25
```

23. random

Syntax: `random min max` **Purpose:** Random number between min and max

```
dice is random 1 6          → Random 1-6
chance is random 1 100      → Random 1-100
```

24. exp

Syntax: `exp number` **Purpose:** e^number

```
result is exp 1             → 2.718...
```

25. ln

Syntax: `ln number` **Purpose:** Natural logarithm

```
result is ln 2.718          → 1
```

26. log

Syntax: `log number` **Purpose:** Base-10 logarithm

```
result is log 100           → 2
result is log 1000          → 3
```

27. factorial

Syntax: `factorial number`

```
result is factorial 5       → 120 (5*4*3*2*1)
result is factorial 3       → 6
```

28. sum

Syntax: `sum list` **Purpose:** Add all numbers in list

```
numbers is [10, 20, 30]
total is sum numbers        → 60
```

29. average

Syntax: `average list`

```
numbers is [10, 20, 30]
avg is average numbers      → 20
```

30. product

Syntax: `product list` **Purpose:** Multiply all numbers

```
numbers is [2, 3, 4]
result is product numbers   → 24
```

CATEGORY 4: CONTROL FLOW (15 COMMANDS)

Conditionals

1. if / when / whenever

Syntax: `when condition { ... }` **Examples:**

```
# Basic condition
score is 100
when score equals 100 {
    say "Perfect score!"
}

# Greater than
health is 75
when health greater 50 {
    say "Healthy"
}
```

```
# Less than
ammo is 5
when ammo less 10 {
    say "Low ammo!"
}

# Not equal
status is "alive"
when status notequals "dead" {
    say "Still alive!"
}
```

2. equals

Purpose: Check equality

```
x is 10
when x equals 10 {
    say "X is 10"
}
```

3. notequals

Purpose: Check inequality

```
status is "playing"
when status notequals "gameover" {
    say "Game continues"
}
```

4. greater

Purpose: Check if greater than

```
score is 1000
when score greater 500 {
    say "High score!"
}
```

5. less

Purpose: Check if less than

```
health is 25
when health less 30 {
    say "Critical health!"
}
```

6. between

Purpose: Check if value is between two numbers

```
temp is 75
when temp between 60 80 {
    say "Comfortable temperature"
}
```

---

7. else / elseif

Syntax:

```
when condition {
    ...
} else {
    ...
}
```

Examples:

```
score is 85

when score greater 90 {
    say "Grade: A"
} elseif score greater 80 {
    say "Grade: B"
} elseif score greater 70 {
    say "Grade: C"
} else {
    say "Grade: F"
}
```

Loops

8. repeat

Syntax: `repeat n times { ... }` **Examples:**

```
# Simple repeat
repeat 5 times {
    say "Hello!"
}

# With counter
count is 1
```

```
repeat 10 times {
    say "Count: " count
    count add 1
}

# Nested loops
repeat 3 times {
    repeat 4 times {
        say "Inner loop"
    }
}
```

9. while

Syntax: `while condition { ... }`

```
count is 1
while count less 11 {
    say count
    count add 1
}

# Infinite loop (use with caution!)
while 1 equals 1 {
    say "Forever"
    # Need break condition
}
```

10. until

Syntax: `until condition { ... }` **Purpose:** Loop until condition becomes true

```
count is 0
until count equals 5 {
    say count
    count add 1
}
```

11. for

Syntax: `for variable from start to end { ... }`

```
for i from 1 to 10 {
    say "Number: " i
}

for x from 0 to 100 {
    when x modulo 10 equals 0 {
        say x
    }
}
```

12. foreach

Syntax: `foreach item in list { ... }`

```
names is ["Alice", "Bob", "Charlie"]
foreach name in names {
    say "Hello, " name "!"
}

numbers is [10, 20, 30, 40]
foreach num in numbers {
    say "Number: " num
}
```

Loop Control

13. break

Purpose: Exit loop early

```
count is 1
repeat 100 times {
    say count
    when count equals 10 {
        break
    }
    count add 1
}
# Only prints 1-10, not 1-100
```

14. continue

Purpose: Skip to next iteration

```
for i from 1 to 10 {
    when i modulo 2 equals 0 {
        continue
    }
    say i  # Only prints odd numbers
}
```

15. return

Purpose: Exit early (in functions)

```
function check_health {
    when health less 0 {
        return
    }
    say "Health OK"
}
```

CATEGORY 5: STRING OPERATIONS (20 COMMANDS)

1. join

Purpose: Combine strings **Syntax:** `join string1 string2`

```
first is "Hello"
second is "World"
result is join first second     → "HelloWorld"
result is join first " " second → "Hello World"
```

2. split

Purpose: Split string into list **Syntax:** `split string delimiter`

```
text is "apple,banana,orange"
fruits is split text ","        → ["apple", "banana", "orange"]

sentence is "Hello World"
words is split sentence " "     → ["Hello", "World"]
```

3. length

Purpose: Get string length **Syntax:** `length string`

```
text is "Hello"
len is length text             → 5

name is "AtlasEngine"
size is length name            → 11
```

4. uppercase

Purpose: Convert to uppercase **Syntax:** `uppercase string`

```
text is "hello"
result is uppercase text        → "HELLO"
```

5. lowercase

Purpose: Convert to lowercase **Syntax:** `lowercase string`

```
text is "HELLO"
result is lowercase text        → "hello"
```

6. titlecase

Purpose: Capitalize first letter of each word **Syntax:** `titlecase string`

```
text is "hello world"
result is titlecase text        → "Hello World"
```

7. trim

Purpose: Remove whitespace from start/end **Syntax:** `trim string`

```
text is "  hello  "
result is trim text             → "hello"
```

8. replace

Purpose: Replace substring **Syntax:** `replace string old new`

```
text is "I like cats"
result is replace text "cats" "dogs"  → "I like dogs"
```

9. substring

Purpose: Extract part of string **Syntax:** `substring string start end`

```
text is "Hello World"
result is substring text 0 5    → "Hello"
result is substring text 6 11   → "World"
```

10. startswith

Purpose: Check if string starts with substring **Syntax:** `startswith string prefix`

```
text is "Hello World"
result is startswith text "Hello"  → 1 (true)
result is startswith text "World"  → 0 (false)
```

11. endswith

Purpose: Check if string ends with substring **Syntax:** `endswith string suffix`

```
text is "Hello World"
result is endswith text "World"    → 1 (true)
result is endswith text "Hello"    → 0 (false)
```

12. contains

Purpose: Check if string contains substring **Syntax:** `contains string substring`

```
text is "Hello World"
result is contains text "Wor"      → 1 (true)
result is contains text "xyz"      → 0 (false)
```

13. indexof

Purpose: Find position of substring **Syntax:** `indexof string substring`

```
text is "Hello World"
pos is indexof text "World"        → 6
pos is indexof text "xyz"          → -1 (not found)
```

14. count

Purpose: Count occurrences of substring **Syntax:** `count string substring`

```
text is "banana"
result is count text "a"           → 3
```

15. reverse

Purpose: Reverse string **Syntax:** `reverse string`

```
text is "Hello"
result is reverse text             → "olleH"
```

16. padleft

Purpose: Pad string on left **Syntax:** `padleft string width char`

```
num is "5"
result is padleft num 3 "0"        → "005"
```

17. padright

Purpose: Pad string on right **Syntax:** `padright string width char`

```
text is "Hi"
result is padright text 5 "."      → "Hi..."
```

18. slice

Purpose: Extract characters (alias for substring) **Syntax:** `slice string start end`

```
text is "Hello World"
result is slice text 0 5           → "Hello"
```

19. pattern

Purpose: Check if string matches pattern (regex) **Syntax:** `pattern string regex`

```
email is "user@email.com"
valid is pattern email ".*@.*\\..*"  → 1 (true)
```

20. convert

Purpose: Convert string to number or vice versa **Syntax:** `convert value type`

```
text is "123"
num is convert text "number"       → 123

number is 456
text is convert number "string"    → "456"
```

TO BE CONTINUED IN PART 3… (Graphics 2D, 3D, Game Mechanics, Physics, and Complete Examples)