AppleScript Cheatsheet: Scripting Mac Automation

Unlock the power of macOS automation with this comprehensive AppleScript cheatsheet covering basic syntax, application scripting, file system operations, UI scripting, and essential tips.

AppleScript Basics & Syntax

Core Syntax Elements

`--` or `#`	Single-line comments.
`(* ... *)`	Multi-line comments.
`set variableName to value`	Variable assignment.
`display dialog "Hello!"`	Show a dialog box.
`say "Hello World" using "Alex"`	Make the computer speak.
`the result`	Get the result of the last command or block.
`return value`	Return a value from a script or handler.
`run`	Explicitly run the main handler of a script.

Data Types

`string`	`"Hello, world!"`
`integer`	`123`
`real`	`3.14`
`boolean`	`true`, `false`
`list`	`{"apple", "banana", "cherry"}`
`record`	`{name:"Alice", age:30}`
`date`	`(current date)`
`alias`	`alias "Macintosh HD:Users:YourUser:file.txt"`
`file`	`file "path/to/file.txt"` (POSIX path)

Control Structures

Conditional Statements:

```
if condition then
    -- Code to execute if condition is true
else if anotherCondition then
    -- Code to execute if anotherCondition is true
else
    -- Code to execute otherwise
end if
```

Repeat Loops (basic):

```
repeat number of times
    -- Code to repeat
end repeat

repeat until condition
    -- Code to repeat until condition is true
end repeat

repeat while condition
    -- Code to repeat while condition is true
end repeat
```

Repeat Loop (iterating over list):

```
set myList to {"a", "b", "c"}
repeat with item from myList
    display dialog item
end repeat
```

Repeat Loop (with counter):

```
repeat with i from 1 to 10
    display dialog "Count: " & i
end repeat
```

Exiting Loops:

```
repeat
    -- Code
    if condition then exit repeat
end repeat
```

Error Handling:

```
try
    -- Code that might cause an error
on error errorMessage number errorNumber
    display dialog "Error: " & errorMessage & " (" & errorNumber & ")"
end try
```

Scripting Applications

Targeting Applications

The `tell` block is used to send commands to a specific application.

```
tell application "Finder"
    -- Commands for Finder go here
end tell

tell application id "com.apple.finder" -- Using bundle ID
    -- Commands for Finder go here
end tell
```

To see available commands for an application, open it in the Script Editor, go to File > Open Dictionary..., and select the application.

Accessing properties within a `tell` block:

```
tell application "Finder"
    set desktopItems to items of desktop
    display dialog "Number of items on desktop: " & (count desktopItems)
end tell
```

Using `whose` or `where` clauses to filter items:

```
tell application "Finder"
    set largeFiles to (files of desktop where size is greater than 1000000) -- size is in bytes
    display dialog "Large files: " & (count largeFiles)
end tell
```

Referring to the current script:

```
tell me
    -- Commands for the current script itself
end tell
```

Referring to the current application (the one running the script, e.g., Script Editor):

```
tell current application
    -- Commands for the host application
end tell
```

Common Finder Commands

`activate`	Brings the application to the front.
`open item`	Opens a file or folder.
`duplicate item to location`	Duplicates an item.
`move item to location`	Moves an item.
`delete item`	Deletes an item (moves to Trash).
`make new folder at location with properties {name:"New Folder"}`	Creates a new folder.
`get properties of item`	Gets a record of an item's properties.
`set name of item to "newName.txt"`	Renames an item.
`exists item`	Checks if an item exists.

Working with Paths

Colon-delimited path	`"Macintosh HD:Users:YourUser:Documents:myfile.txt"` (Old HFS path - still used by `alias`)
Slash-delimited path	`"/Users/YourUser/Documents/myfile.txt"` (POSIX path)
`path to folder`	Get standard folder paths (e.g., `path to desktop folder as alias`)
`alias "path:"`	Reference an item by its HFS path.
`(POSIX file "path") as alias`	Convert POSIX path to alias.
`(the alias as POSIX path)`	Convert alias to POSIX path.
`quoted form of POSIX path`	Escape spaces and special characters for use in shell scripts.

UI Scripting & System Events

System Events Application

`System Events` is the main application for interacting with the macOS UI and internal processes.

```
tell application "System Events"
    -- Commands for UI scripting or system processes
end tell
```

Enable 'Enable access for assistive devices' or grant permissions under Security & Privacy > Accessibility in System Settings for UI scripting to work.

Inspecting UI Elements: Use the 'Accessibility Inspector' developer tool (or third-party tools like UI Browser) to find names/IDs of UI elements (windows, buttons, text fields, etc.).

Example: Click a button in a specific window:

```
tell application "System Events"
    tell process "Finder"
        tell window "Downloads"
            click button "Close"
        end tell
    end tell
end tell
```

Example: Set value of a text field:

```
tell application "System Events"
    tell process "TextEdit"
        tell window 1
            tell text area 1 of scroll area 1
                set value to "This is some text."
            end tell
        end tell
    end tell
end tell
```

Example: Get list of running processes:

```
tell application "System Events"
    set runningProcesses to name of every process whose
background only is false
    display dialog (runningProcesses as string)
end tell
```

Keyboard Interaction

`keystroke "string"`	Types the specified string.
`key code number`	Sends a key press by its key code.
`keystroke "a" using {command down}`	Sends a keystroke with modifier keys (command, shift, option, control).
`key code 36 using {command down, option down}`	Sends key code with multiple modifiers (e.g., Command+Option+Return)
Common Key Codes	Return: 36, Tab: 48, Space: 49, Escape: 53, Left Arrow: 123, Right Arrow: 124, Down Arrow: 125, Up Arrow: 126

Running Shell Scripts

Execute shell commands directly.

```
set shellScript to "ls -l ~/Documents"
set scriptResult to do shell script shellScript
display dialog scriptResult
```

Running with administrator privileges:

```
set shellScript to "rm -rf /some/sensitive/path"
display dialog "Execute command?" buttons {"Cancel", "OK"}
default button "OK"
if the button returned of the result is "OK" then
    try
        do shell script shellScript with administrator
privileges password "YourPasswordHere" -- Not recommended to
hardcode password!
    on error errMsg
        display dialog "Shell script failed: " & errMsg
    end try
end if
```

Note: Hardcoding passwords is a security risk. Consider using `with administrator privileges` and letting the system prompt the user.

Handling paths and special characters in shell scripts requires careful quoting. Use `quoted form of POSIX path`.

```
set myFile to path to desktop as string
set myFilePOSIX to POSIX path of myFile
set quotedFile to quoted form of myFilePOSIX
set shellScript to "open " & quotedFile
do shell script shellScript
```

Advanced Topics & Tips

Handlers (Functions/Subroutines)

Define reusable blocks of code.

```
on sayHello(userName)
    display dialog "Hello, " & userName & "!"
end sayHello

-- Calling the handler
sayHello("Alice")
```

Handlers with multiple parameters:

```
on addNumbers(num1, num2)
    return num1 + num2
end addNumbers

set sumResult to addNumbers(5, 7)
display dialog "Sum is: " & sumResult
```

Handlers with labeled parameters:

```
on greet name userName andAge userAge
    display dialog "Hello " & userName & ", you are " & userAge
& " years old."
end greet

-- Calling with labels
greet name "Bob" andAge 25
```

Working with Lists & Records

Accessing list items	`set myList to {"a", "b", "c"}`
	`set firstItem to item 1 of myList`
	`set lastItem to last item of myList`
	`set middleItem to item 2 of myList`

Counting items	`set itemCount to count myList`
Adding items	`set myList to myList & {"d"}`
Converting list to string	`set listString to myList as string`

Accessing record properties	`set myRecord to {name:"Charlie", city:"London"}`
	`set userName to name of myRecord`
	`set userCity to myRecord's city`

Checking if record has property	`if myRecord contains {city:missing value} then ...`

Useful Commands & Tips

`delay seconds`	Pause script execution (e.g., `delay 1.5`). Useful for UI scripting.
`clipboard info`	Get information about the clipboard content.
`the clipboard as datatype`	Get clipboard content as text, alias list, etc. (e.g., `the clipboard as text`)
`set the clipboard to value`	Set clipboard content.
`quoted form of string`	Escapes special characters in a string for shell scripts.
`text item delimiters`	Change the delimiter used by `text items` (e.g., `set AppleScript's text item delimiters to ", "`). Remember to reset!
Best Practice: Save as Application	Saving a script as an application (`.app`) allows it to be run directly or via keyboard shortcuts (using Automator) or login items.
Best Practice: Stay in Scope	Use nested `tell` blocks to keep commands specific to their target application/element.
Tip: Debugging	Use `display dialog` to show variable values or script progress during execution. The 'Result' pane in Script Editor shows the last returned value.