

## Introduction

(<https://livecode.com/docs/9-5-0/introduction/>)

## Lessons

(<https://livecode.com/docs/9-5-0/lessons/>)

FAQ (<https://livecode.com/docs/9-5-0/faq/>)

## Language

(<https://livecode.com/docs/9-5-0/language/>)

LiveCode Cheat Sheet

(<https://livecode.com/docs/9-5-0/language/livecode-cheat-sheet/>)

LiveCode Builder Cheat Sheet

(<https://livecode.com/docs/9-5-0/language/livecode-builder-cheat-sheet/>)

LiveCode Builder - LiveCode Cheat Sheet (<https://livecode.com/docs/9-5-0/language/livecode-builder-livecode-cheat-sheet/>)

LiveCode Script

(<https://livecode.com/docs/9-5-0/language/livecode-script/>)

## Education Curriculum

(<https://livecode.com/docs/9-5-0/education-curriculum/>)

## Deployment

(<https://livecode.com/docs/9-5-0/deployment/>)

## Components

(<https://livecode.com/docs/9-5-0/components/>)

## Tooling

(<https://livecode.com/docs/9-5-0/>)

# LiveCode Builder Cheat Sheet

## Comments

Comments allow you to add explanations and annotations to your code.

```
-- these  
// are  
/* commented  
out */
```

## Literals

A literal is a notation for creating a particular type of value.

```
"string literal"  
["list", "of",  
 "literals"]  
{ "array":  
  "literal" }
```

## Variables

Comments

Literals

Variables

Constants

Control  
Structures

Operators

String  
Processing

Array  
Processing

Sorting

Files &  
Processes

Custom  
Handlers

Event  
Handlers

0/tooling/)

Core Concepts

(<https://livecode.com/docs/9-5-0/core-concepts/>)

Language Comparison

(<https://livecode.com/docs/9-5-0/language-comparison/>)

Extending LiveCode

(<https://livecode.com/docs/9-5-0/extending-livecode/>)

Whats New?

(<https://livecode.com/docs/9-5-0/whats-new/>)

Variables are used to to store information, the stored value can be changed or accessed when you need it.

```
variable tVar  
put "str" into tVar  
put 1 into tVar
```

```
variable tArr as  
Array  
put "val" into  
tArr["key"]
```

## Constants

Constants store a value that is defined at the point of declaration and never changes.

```
constant kFoo is  
15
```

## Control Structures

Control structures are used to control what code is executed and how many times.

```
repeat for each  
char tChar in tVar  
end repeat
```

```
repeat 10 times  
end repeat
```

```
repeat with tX  
from 1 up to 10  
end repeat
```

```
repeat while tX >  
1  
    subtract 1  
from tX  
end repeat
```

```
if tVar then  
else if tOther  
then  
else  
end if
```

## Operators

Operators are ways of combining values such as boolean values, numbers or strings, to produce other values.

```
// Logical
true and false is
false
true or false is
true
not false is true

// String
"foo" & "bar" is
"foobar"
"foo" && "bar" is
"foo bar"
"string" begins
with "st"
"string" ends with
"g"

// Chunks
char 5 of "string"
is "n"

split "a,b,c" by
"," into tItems
tItems[3] is "c"

split "hi there" by
" " into tWords
tWords[1] is "hi"

split "anb" by "n"
into tLines
tLines[2] is "b"

split "a,b,c" by
"n" into tLines
split tLines by ","
into tItems
char 1 of tItems[1]
is "a"
```

## String Processing

These examples show how string values can be manipulated.

```
// General
put "a" before tVar
delete char 1 of
tVar
replace "_" with
"-" in tVar
```

## Array Processing

These examples show how array values can be manipulated.

```
// Split / combine
put "a,b,c" into
tVar
split tVar by ","
tVar[2] is "b"
combine tVar with
","
tVar is "a,b,c"
```

```
// Iteration
repeat for each
key tKey in tArray
  -- Do something
with tArray[tKey]
end repeat
```

```
repeat for each
element tElement in
tArray
end repeat
```

```
// Length
the number of
elements in tArray
```

## Sorting

These examples show how to sort items and lists.

```

variable tList
put [5,2,3,1,4]
into tList
sort tlist in
ascending numeric
order
-> tList is
[1,2,3,4,5]
sort tlist in
descending numeric
order
-> tlist is
[5,4,3,2,1]

public handler
DoSort(in pLeft, in
pRight) returns
Integer
    return
pLeft[2] -
pRight[2]
end handler

```

```

variable tData as
List
put [[6, 1], [8,
3], [2, 2]] into
tData
sort tData using
handler DoSort
-> tData is [[6,
1], [2, 2], [8, 3]]

```

## Files & Processes

These examples show how to read from and write to files and processes.

```

get the contents of
file tPath
set the contents of
file tPath to ""

```

## Custom Handlers

A custom handler is a function or command that you define yourself.

```
handler foo(in
pParam)
end foo
// get foo(tVar)
// foo 5
```

## Event Handlers

An event handler is a handler that is triggered when an event occurs, such as the use of the mouse or keyboard.

```
// Mouse
handler OnMouseUp()
  get the click
  button
end handler

handler
OnMouseDown()
  get the click
  button
end handler

handler
OnMouseMove()
end handler

// Keyboard
handler
OnKeyPress(in
pText)
end handler
```

