Python - LiveCode Builder Cheat Sheet

Comments

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

Python

```
# this is
commented
out
```

LiveCode Builder

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

Literals

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

Python

LiveCode Builder

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

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

Variables

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

Python

```
var = "str"
var = 1

var["key"] = "val"
```

LiveCode Builder

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

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

Control Structures

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

Python

```
for x in tVar:
    # do things
for x in
range(10):

do
things


while x > 1:
x -= 1
```

```
if tVar:
elif tOther:
else:
```

LiveCode Builder

```
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.

Python

```
# Logical
true and false == false
true or false == true
!false == true
```

String

```
"foo" + "bar" == "foobar"
strs = ['foo','bar']
' '.join(strs) == "foo
bar"
"string".startswith("st")
"string".endswith("g")
```

Chunks

```
"string"[4:5] == "n"


items =
"a,b,c".split(",")
items[2] == "c"


words = "hi
there".split(" ")
words[0] == "hi"


lines = "anb".split("n")
lines[1] == "b"
```

```
lines = "a,b,c".split("n")
items = lines[1].split(",")
items[1][0:1] == "a"
```

LiveCode Builder

```
//
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.

Python

LiveCode Builder

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

```
# General
var = 'a' + var
var = var[1:]
var.replace("_",
"-")
```

Regex

```
found =
re.match('([0-
9])', '1')
num =
tMatch.group(1)
```

```
for line in var:
if re.match(pattern,
line):
filtered.push(line)
var =
'n'.join(filtered)
```

Array Processing

These examples show how array values can be manipulated.

Python LiveCode B

```
# Split / combine
var = "a,b,c".split(",")
var[1] is "b"
','.join(var)
var == "a,b,c"
```

Iteration

```
for key in array:
    do something with array[key]
```

Length

```
len(array)
```

```
// Spli
combin
put "a
into tV
split
by ","
tVar[2]
"b"
combin
with ",
tVar is
"a,b,c"
// Ite
repeat
each ke
in tArr
-- Do
someth
with
tArray
end re

repeat
each el
tElemen
tArray
end re
```

```
// Length
the numbe
elements i
```

Sorting

These examples show how to sort items and lists.

Python LiveCode Builder

```
list = [5, 2,
3, 1, 4]
sorted(list)
== [1, 2, 3,
4, 5]
sorted(list,
reverse=True)
== [5, 4, 3,
2, 1]
```

```
data = [(6, 1),
(8, 3), (2, 2)]
sorted(data,
key=itemgetter(2))
== [(6, 1), (2,
2), (8, 3)]
```

```
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.

Python

```
open(tPath).read()
open(tPath).write("")
```

```
process =
subprocess.Popen([tProc],
stdout=subprocess.PIPE)
while True:
process.wait()
data =
process.stdout.read(5)
if data:
break
```

LiveCode Builder

```
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.

Python

```
def
foo(param):
# return
something
#
foo(var)
```

LiveCode Builder

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