

Catscript Guide

The following documentation addresses the Catscript programming language and the syntax necessary to code in the functional language. The Catscript compiler is based on a Java transpiled system that outputs through the JVM Bytecode systems, but is centralized to use just Catscript coding components listed below.

Introduction

Catscript is a simply functional language that works with functions, expressions, statemnets, variables, and other syntax components familiar to Java, Python, and other functional components in most languages. The following is an example of the base look and coding of the Catscript language:

```
var x = "foo"
print(x)

if(x=="foo")
{
    print("truth")
}
else
{
    print("false")
}
```

The language is statically and strongly typed, allowing us to mimic some of the basic type functionality of Javascript and Python langauges. The following documentation will address the usage of expressions, statements, functions, types and variables, and the syntax necessary to produce error-free code with Catscript.

Example of the Catscript Language

- ♦ Examples: Expressions and Statements

```
var x : int = 10
var arithVar : int = (x + 1) / (4 * x)
var y : string = "hello"
var listDemo : list<int> = [1, 4, 5, 6, 7]

var z = true // Assumes the Type
var nullType : object = null

print(x) // Prints Variable
print(y)

if (z == true) // Determines if Z equals true, boolean
equalities.
{
    z = false
```

```
}
else if (arithVar >= 1)    // Compares arithmVar to see if it is greater
                           than or equal to one, comparisons.
{
    print(nullType)
}
else
{
    print(y)
}

for(iterator in [1, 2, 3, 4])    // Iterates over a list of 4 values,
                                prints the values and parts of our list, listDemo.
{
    print(iterator)

    print(listDemo[iterator])
}
```

- ♦ Examples: Functions

```
function foo() {                                // function foo() executes a
printing of added variables

    var foo_x : int = 0
    var foo_y : int = 10

    var foo_z = 0 + 10

    print(foo_z)
}

function bar(a: int, b: int) : bool {          // function bar() returns a
boolean, decided by checking if a and b equal 10

    var output : bool = null
    var total : int = (a + b)

    if (total == 10)
    {
        output = true
    }
    else
    {
        output = false
    }

    return output
}

print(foo())    // Will Print 10
```

```
print(bar(5, 5))    // Will Print TRUE
```

Programming in Catscript

- ♦ Types and Variables

Types

The following topic covers the types that are usable within the Catscript language, which will hold similar typing as most common languages. The types used in Catscript are as follows: **INT**, **STRING**, **BOOL**, **LIST**, **NULL**, and **OBJECT**. The types are used to define *variables* and *functions* that will be used within the language. Types will be demonstrated below in several examples, the statements, variables, and functions will be defined further on.

```
expression - statement NO_TYPE ;
expression - statement : int ... ;
expression - statement : string ... ;
expression - statement : bool ... ;
expression - statement : object ... ;
expression - statement : list [... : TYPE ] ... ;

expression - statement : ANY_TYPE -> set to NULL ;

function "name" ( ... ) : TYPE { ... } ;
```

Variables

The following topic covers the variables in Catscript and how they are created to be used in functions, statements, and with expressions. The variables will have types and will be demonstrated in the examples below.

```
var value = TYPE_VALUE ;
var value : TYPE = TYPE_VALUE ;

var number : int = 0 ;
var word : string = "string word" ;
var truthy : bool = true ;
var listy : list<TYPE> = [val1, val2, val3, ...]
var random : object = VALUE ;

var nullEx = null ;
var nulledType : TYPE = null ;
```

- ♦ Expressions

Additive, Subtractive, Multiplied, Divided

The following examples demonstrate the different forms of arithmetic capabilities of the Catscript language. The language is capable of adding, multiplying, subtracting, and dividing integers. Additionally, Catscript can concat strings, nulls, and some objects with the right typing. We also see how parenthesis are capable of bring priority to arithmetic values and equations. The following information is demonstrated in the examples below.

```
val1 + val2, var total = val1 + val2, var total : STRING_INT = (val1 + val2)
OR val1 + val2 [TYPES MUST MATCH VAR TYPE]
val1 - val2, var total = val1 - val2, var total : INT = (val1 - val2) OR
val1 - val2
val1 * val2, var total = val1 * val2, var total : INT = (val1 * val2) OR
val1 * val2
val1 / val2, var total = val1 / val2, var total : INT = (val1 / val2) OR
val1 / val2
```

EX

```
var endString : string = "word"
var number_value : int = 1
```

```
var string_concat : string = "String " + endString ->      "String word"
var string_int_cat : string = "Value: " + number_value ->   "Value: 1"
var string_null_cat : string = endString + null ->          "wordnull"
```

```
var int_add = 10 + 100 ->      110
var int_minus = 5 - number_value ->  4
var int_multi = 10 * 10 ->      100
var int_divis = 50 / number_value ->  50
```

```
var long_arithm = 5 + 10 * (number_value / 5) ->  "3 = 15 / 5"
```

Boolean, Null

The following example shows the use case of Boolean and Null values. The use of Booleans are often used in choices and systems of checking within a function or output. The use of Nulls can be used to return certain outputs in failed cases and is often used as a placeholder for systems. The following information is demonstrated in the examples below.

```
true, false, var bool_value = true, var bool_value = false
```

```
null, var null_value = null, var null_value : ANY_TYPE = null
```

EX

```
var big_value : int = null      // Can be used to predefine values before we
use them, so we do not accidentally trigger choices
var val : bool = true          // Set variables to a boolean value when
their TYPE is bool
```

```

var ex_val = false           // Assumes the boolean type value

if (val == null)             // NULLS can be used as checks in choice structure
and error structure
{
    ... body ...             // We could return a function if our value is NULL,
to protect the program
}

if (val == true)             // checks if a variable or literal is equal to the
boolean true
{
    ... body ...
}

if (val == false)           /// checks if a variable or literal is equal to the
boolean false
{
    ... body ...
}

```

Comparisons, Equivalence

The following example shows the use case of Comparisons and Equivalence in the Catscript language. Comparisons are essential to making logical pathed decisions in the Catscript compiler environment, while Equivalence is how most of our choice structure works in the language. We can see below how equivalence allows us to make coordinate choices and how it can be used alongside comparisons to become even more specific with our choice structures.

```

val1 == val2, val1 != val2

val1 < val2, val1 <= val2
val1 > val2, val1 >= val2

EX
var val1 : int = 1
var val2 : int = 5
var bool_val : bool = true

if (bool_val == false)       // IF values are equal then its true
{
    ... true body ...
}

if (bool_val != false)       // IF values are not equal then its true
{
    ... true body ...
}

```

```

if (val1 >= val2)           // IF values are greater than or equal to second
values then its true
{
    ... true body ...
}

if (val1 < val2)           // IF values are less than to second values then
its true
{
    ... true body ...
}

```

- ♦ Statements

If Else Statements

The following example shows the use of If Statements and how to make proper choices with variables, literal type values, and other systems to build complex deciding structures. This system of statements allow an individual to code choices into their program, below will show examples of the following statements.

```

if (expression) {
    ... true body ...
}
else {
    ... false body ...
}

if (expression) {
    ... 1st true body ...
}
else if (expression) {
    ... 2nd true body ...
}
else if ... {
    ... nth true body ...
}...
else {
    ... false body ...
}

EX
if (val1 == val2)
{
    print(val1)           // Prints val1 if values are equal, and val2 if they
are not.
}
else

```

```

{
    print(val2)
}

if (x == 1)          // Switch-like system of executing body expressions
when conditions are true in the if.
{
    print(x)
    bool_value = true
}
else if (x > 5)
{
    print("Bigger")
}
else if (x > 100)
{
    print(100000)
    x = 100000
}
else
{
    print("poor")
    x = 0
    bool_value = false
}

```

For Statements

The following example shows the use of For Statements and how to make looping systems with our Catscript data structures. The system will use variable structures to help iterator over the loop, which can use global and local scoped items.

```

for (variable in [val1, val2, val3, ...]) {
    ... body ...
}

EX
for ( x in [1, 2, 3] ) {           // Iterate X over the literal list, [1,
2, 3]

    print(x)

    if(x == 1)
    {
        print("Start")           // Print structures can take literal
values
    }
}

```

Functions Header

The following emphasizes the header and how it is used to initialize a function in Catscript. The function header is necessary for building the main structure identified by our Catscript compiler.

```
function funcName (...) {... body ...}

EX
function fooBar (...) {... body ...}
```

Functions Parameters and Variables

The following emphasizes the parameters and variables that are implemented into the header of the function. The parameters and variables must be defined, along with the function type, to use variables that are necessary to our functions execution. Not all functions will have the parameters filled in, or the type defined.

```
function funcHeader (parmA : TYPE, paramB : TYPE) : Func_TYPE {... body ...}

EX
function fooBar (value_A : int, value_X : bool) : int {... body ...}

function fooBar (NO PARAM) : NO_TYPE {... body ...}
```

Functions Body

The following emphasizes the function body data structure, which holds the specific processing and combination of data structures within Catscript that an individual can use. The body will return values, using a return statement, and output other values if not return is specified.

Below the example will showcase all of the following components of a Function and its internal data structures.

```
function funcHead (parmA : TYPE, paramB : TYPE) : Func_TYPE {

    body_variables OR body_expressions

    body_statements

    return_statement --> REQUIRED IF TYPE DEFINED
}

function funcHead (NO PARAM) : NO_TYPE {

    body_variables OR body_expressions
```



```

    body_statements

    return_statement --> OPTIONAL IF TYPE NOT DEFINED
}

EX
function helper (key : int, value : string) : bool {

    var key_verified : bool = null

    if (key == 1)                                // Key System to Verify Value state
    {
        value = "output is 1"
        key_verified = true
    }
    else if (key == 2)
    {
        value = "output is 2"
        key_verified = true
    }
    else if (key == 3)
    {
        value = "output is 3"
        key_verified = true
    }
    else
    {
        key_verified = false
    }

    return key_verified                          // Returns Checked State
}

function execution (adder : int) {

    var multiply_Key : int = 5
    var new_value = ( adder * multiply_Key)      // Type is Assumed

    return new_value
}

function cat (newEnd : string) {

    var phrase : string = "Hello, my name is "
    var new_phrase : string = phrase + newEnd   // Catscript concantentation
of Strings

    print(new_phrase)                           // No Return is Needed, The
function just executes when called.
}

cat("dog")                                     // Takes a string and prints out a phrase,

```

```
execution function.  
print(execution(1))          // Function returns a value that can be set to a  
variable or printed.  
var checker : bool = helper(1, stringValue)    // Function returns boolean  
and updates a string variable we enter into the funtion
```