Catscript Guide

Introduction

Catscript is a simple scripting langauge. Here is an example:

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

Features

Expressions

Additive Expression

Operators:

- `+` Addition
- `-` Subtraction

The additive expression illustrates the use of using the + operator to add integers together.
It can also be used to concatenate integers to strings, strings to strings, and strings to null values
For integers, this also includes the - symbol as well. The subtraction operator subtracts the right-hand side of the expression from the left-hand side of the expression.

```
// addition examples
print(3+4)      // 7
print(3+"a")    // 3a
print("a"+null) // anull

// subtraction example
print(3-2)      // 1
```

Boolean Literal Expression

The Boolean literal expression represents the logical boolean datatype. Booleans can be one of two values: `true` and `false`.

```
var a = true
var b = false
print(a) //  true
print(b) //  false
```

Comparison Expression

Operators:

- `<` Less Than
- `>` Greater Than
- `<=` Less Than or equal to
- `>=` Greater Than or Equal to

The purpose of these operators is to perform a logical comparison against two expressions. Comparison expression return a boolean value. If the comparison is correct, it will return `true`. If not, then the expression will evaluate to `false`.

```
//integer examples
print(2<1)  // false
print(2>1)  // true
print(2<=1) // false
print(2>=1) // true
```

Equality Expression

Operators:

- `==` Equal to
- `!=` Not Equal to

You can check equality from integers to integers, booleans to booleans, and strings to strings. The expression evaluation will return a boolean value. `true` will be returned if the equality is true, otherwise a `false` will be returned.

```
//integer examples
print(1==1) // true
print(1!=1) // false

//boolean examples
print(true!=true)   // false
print(false==false) // true

//string examples
print("a"=="a") // true
print("a"=="b") // false
print("a"!="b") // true
```

Factor Expression

Operators:

- `*` Multiplication
- `/` Division

The multiplication operator is used to multiply numerical expressions to produce a product. Similarly, the division operator is used to divide numerical expressions to produce a quotient.

```
print(4*5) // 20
print(20/5) //4
```

Function Call Expression

Function calls are used to execute an already-defined function. If the function definition has a return value, the function call will become the returned value once the function is executed.

In the example below, assume **foo** (which returns an integer) and **bar** (which returns void) has already been defined. The code is calling **foo** that consists of three parameters and assigns the returned value to the variable `x`.**bar** is called with zero parameters and is not assigned to a variable.

```
var x = foo(1, 2, 3)
bar()
```

Integer Literal Expression

An integer literal expression describes the use explicit placement of the `integer` data type.
You can assign literals to a variable, or execute a function with them directly.

```
var x = 5
print(x)    // 5
print(4)    // 4
```

List Literal Expression

A list Literal expression describes the use explicit placement of the `listType` data type.
You can assign literals to a variable, or execute a function with them directly.

```
var x = [1, 2, 3]
print(x)                    // [1, 2, 3]
print([true, false, false]) // [true, false, false]
print(["a", "b", "c"])      // [a, b, c]
```

Null Literal Expression

A null literal expression describes the use explicit placement of the `null` data type.
You can assign literals to a variable, or execute a function with them directly.

```
var x = null
print(x)
print(null) // null
```

Parenthesized Expression

Parenthesized expressions set priority on whatever is within the left and right parentheses. This is useful for grouping and setting precedence over an order of operations.

```
print((4+3)*3) // 21
```

String Literal Expression

A string literal expression describes the use explicit placement of the `string` data type.
You can assign literals to a variable, or execute a function with them directly.

```
var x = "Testing123"
print(x)                // Testing123
print("Hello world!")   // Hello world!
```

Unary Expression

Operators:

- `not` Negation
- `-` Negative

The negation operator is used to inverse the logical boolean of an expression. It will evaluate `true` booleans as `false` and `false` booleans as `true`. The negative operator is used to flip a positive number to a negative number, and negative numbers to a positive number.

```
// Negative
print(-5) // -5

//Negation
print(not true) // false
```

Statements

Assignment

The assignment statement is used to update the value of a predefined variable. It is necessary for the variable type and the new value to be compatible types, an **Incompatible types** error will be thrown if they aren't compatible.

```
// Legal:
var x = 3
x = 4 // Assignment aspect, the value of x has been updated to 4

// Illegal:
var x = 3
x = true // ERROR: Incompatible types
```

For loops

The for-loop assignment is used to iterate over every object within a list. The function will loop once for every item within the list, then assign it to the temporary variable the user defines.

```
// 'letter' is the temporary iterative variable
for(letter in ["a", "b", "c"]){
    print(letter)
}

// 'i' is the temporary iterative variable
var x = [true, false, false]
for(i in x){
    print(i)
}
```

Function Definition

The function definition statement is how you define new functions that you can call on to execute. Function definitions do not execute unless called on. If the function does not have an explicit return type, then the return type will be assigned to `VOID`, which means the function will expect no return.

This first example below demonstrates a function with no parameters and no return type.

```
function sayHello(){
    print("hello!") // function body here
}
```

This function below is called 'myFunction' and it takes two parameters. Since parameter typing is optional, the first parameter type 'one' is not declared, but the second parameter 'two' is declared as a string, therefore, a string must be provided to the parameter 'two'.

This function is told to return a string type through the ': string' after the parameters have been declared. Once a return type is declared, the function must have a return statement.

```
function myFunction(one, two:string) : string {
    // body statements below:
    print(one)
    print(two)
    return two
}
```

Function Calling

Function calling is used to call a predefined function. By calling a function and inserting the parameters (if any), the function will execute the lines of code with the function's body. Examples using the functions defined above: `sayHello()` and `myFunction()`.

Executing function with no parameters and no return:

```
//function definition
function sayHello(){
    print("hello!") // function body here
}
```

```
//function call
sayHello()  // "Hello!"
```

Storing function call's return into Variable:

```
// function definition
function myFunction(one, two:string) : string {
    // body statements below:
    print(one)
    print(two)
    return two
}

//storing the result of the function call into variable:
var result = myFunction(15, "my String")

//print output
print(result) // "my String"
```

If Statements

If statements are used to execute unique lines of code if and only if the expression within the parenthesizes is true. If they are not true and there is an else statement after, then the else statement will execute.

In the code below, true is always true, so the code `print(1)` will execute, but `print(2)` will not.

```
if(true){
    print(1) //will execute
} else {
    print(2) //will not execute
}
```

Since the result of 1==x is false (since 1 is not equal to 2), the if statement is false. Therefore, the lines of code within the if statement will never be reached. The code will carry onto the `else if` statement, which is true since x has the value of 2, and 2 is equal to 2. `print(4)` is

executed. If the `else if` statement was also false, then every line of code within the `else` statement's body will be executed.

```
var x : int = 2
if(1==x){
    print(1)  // Never reached
    print(2)  // Never reached
    print(3)  // Never reached
} else if(2==x){
    print(4) // 4
} else{
    print(5)
}
```

Print Statements

The print statement is used to output an expression's value into the console.

```
print("Printing works") //output: "Printing works"
print(4+5)              // output: 9
```

Return Statements

The return statements are used within function definitions to return a specific value from within the function. The function's return type is defined by the `: int` after the parameter list is finished being defined.

```
// function definition that returns the string data type
function myFunction(one, two:string) : int {
    // body statements below:
    print(one)
    print(two)
    return 40
}

//storing the result of the function call into variable:
```

```
var result = myFunction(15, "my String")

//resul's value:
print(result) // 40
```

Variable Statements

Variable statements are used to define a brand new variable. you can define a new variable with implicit typing with `var VARNAME = VALUE`, or explicitely define a type with `var VARNAME : DATATYPE = VALUE`. Every word in capital letters is a placeholder.

Defining variables implicit typing

```
var a = 4
var b = "hello"
var c = true
```

Defining variables explicit typing

```
var x : int = 4
var y : string = "hello"
var z : boolean = true
```