

Catscript Guide

Introduction

Catscript is a toy programming language compiled by a custom tokenizer, parser, and evaluator created in MSU's CSCI 468 class. It is a statically typed language, offers a small type system, and includes logic for functions, loops, and conditional logic.

Features

CatScript Typing

Booleans

True and false represent the logical `true` and `false` respectively.

```
true // evaluates to logical true
false // evaluates to logical false
```

Integers

The 32 bit signed `integer` type.

```
-100
0
100
```

Strings

A UTF-8-encoded, growable string. Supports string concatenation.

Examples

You can create a `string` using the "word" syntax.

```
"Hello"
```

You can concatenate strings using the `+` operator.

```
"Hello World" == "Hello " + "World" // evaluates to true
```

Null

Conventional null type. Used when a variable is not instantiated or set to another type.

```
null
```

Examples

You can set a value to null, and conditionally check if a variable is null. The null type does not support addition or any other operation.

```
var x = null
(x == null) // evaluates to true
print(x + 1) // incompatible types exception
```

Object

Wraps a primitive type to abstract type information. The common use case is when there is a `List` of `objects`.

```
var my_list: list<object> = ["hello", 3, null]
```

Void

The default return type of a function that does not explicitly return something. Returns the equivalent of nothing. Note, this is different from null.

```
function foo() {} // returns nothing (also know as void)
function add_one(x: int) : int { return x + 1 } // returns and integer (x+1)
```

Lists

An immutable collection of the same type.

```
[1,2,3] // list literal expression
var my_string_list: list<string> = ["hello", "world"] // statement with list type
var my_object_list: list<object> = [null, 3, "foo"]
```

CatScript Operations

Adding

Add two factor expressions together with the binary `+` operator.

```
1 + 1 // evaluates to 2
4 * 5 + 3 // evaluates to (4*5)+3 -> 20+3 -> 23
```

Subtracting

Subtract two factor expressions together with the binary `-` operator.

```
1 - 1 // evaluates to 0
(4*5) - 3 // evaluates to (4*5)-3 -> 20-3 -> 17
```

Logical

Compare two expressions using the `&&` or `||` operator.

```
true && false // evaluates to false
true && true // evaluates to true
true || false // evaluates to true
false || false // evaluates to false
```

Multiplication

Multiply two logical expressions together with binary `*` operator.

```
1 * 8 // evaluates to 8
1 * 2 * 3 // evaluates to 6
```

Division

Divide two logical expressions together with binary `/` operator.

```
6 / 3 // evaluates to 2statements
6 / 4 // evaluates to 1
6 / 10 // evaluates to 0
```

For loops

Iterate through a collection of elements and execute one or more statements each loop.

Examples

A common use case is iterating over a list.

```
var foo = [1,2,3]
for (number in foo) {
    print(number)
}
```

output: 1 2 3

If Statements

Branching logical if-else statements similar to other languages. Each condition is followed by a block. If a condition evaluates to `true`, the statement block directly following is executed and the other conditions are not evaluated.

Examples

The conditions in the if-else branch are only evaluated up to the first condition that evaluates to `true`. If no condition is evaluated to `true`, the `else` statement block will execute.

```
var x = 2
if (x == 1) {
    print("x is 1")
}
else if (x == 2) {
    print("x is 2")
}
else {
    print("x is NOT one or 2")
}
```

output: x is 1

When multiple `if` statements are in succession, each will be evaluated separately. Each starting `if` statement is treated as the start of a new if-else branch.

```
var x = 1
if (x == 1) {
    print("x is 1")
}
if (x != 2) {
    print("x is not 2")
}
```

output: x is 1 x is not 2

Print Statements

A conventional print statement is used to output information to the user in the console. It is an example of a CatScript built in function.

```
print("Hello CatScript") // executes to output "Hello Catscript"
```

Functions

The `function` keyword lets you define a function `identifier`, optional parameters, an optional return type, and a function body. Function bodies encapsulate logic and enter a new scope. When referencing variables or data, the current scope will be searched first. See `Returning Values` for examples and information on returning values from functions.

Examples

Functions can have no parameters.

```
function foo() {}
```

Functions can have parameters.

```
function add_one(number) {  
    number + 1  
}
```

Functions can have parameters with explicit types.

```
function add_one(number: int) {  
    number + 1  
}
```

Function Calls

Functions are called using two parentheses following the function identifier.

```
function foo() {}  
  
foo() // the code inside the foo function block executes
```

Returning Values

Functions can return void or they can return an explicit type. The `return` keyword is required when an explicit return type is defined in the function definition. If no explicit type is defined, `void` is returned.

```
function add_one(number: int) {  
    number + 1  
}  
function add_one(number: int): int {  
    return number + 1  
}
```

Variables

Data can be captured using variables. Every variable has an identifier and an assigned value. The value is of any type in the CatScript programming language. Variables are defined using the `var` keyword.

```
var x = 10  
var y = "foo"  
var z = null
```

Assignment

Variables can be assigned to other pieces of data using the `=` operator. The considered variable must be assignable from the considered value.

```
var x = 10  
x = 20  
x = "foo" // incompatible types error
```