

Section 4: Technical writing

Introduction

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

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

General Syntax

Semicolons are not used to denote the end of the line. Instead, instructions need to be separated with white space. This can be a new line as well.

```
print(x) // new line separates this instruction from the next
print(y)

pring(x) print(y) // white space separated these instructions from one another
```

Comments are started with the slash slash "//", and ended with a new line. * Comments are parts of the code that is not compiled * // this is a comment that won't be interacted with * Note that there are no multiline comments in Catscript.

Variables

A variable is a container that holds data. Each variable has a specific data-type. Variables must first be initialized. Then the variable can be set to a new value or used to access the data it holds. Note variables cannot be declared. Instead, they must always hold a value.

Types

The data-type of a variable controls what can be stored, and how it behaves within expressions. * Integers * An integer is a whole number (no decimals). * Strings * A string holds a sequence of characters. * Booleans * A Boolean holds either true or false. * Objects * Can represent any variable data-type, since it is a wrapper type. * Lists * A List is a group of another data type. * A List of objects can store multiple different types.

Initialization

Before a variable can be interacted with, it must be initialized. In which it's data-type is set, and it is given data to hold. Variables can be set implicitly or explicitly. In the following examples, the first is implicit and the second is explicit.

To initialize a variable, the keyword "var" is followed by the variable name. Then by an equal sign, and the value it is being set to. * Variable names must start with either a letter or an underscore "_". * Variable names must be unique. Two variables cannot exist at the same time (see scoping) with the same name.

```
var x = 1
```

Implicit setting

Variables can be initialized implicitly, in which no data-type is defined. Instead, the data-type is assumed from what is being stored.

Explicit setting

Variables can have explicit setting as well. This is when after the name of the variable, there is a colon ":" followed by the keyword for that data type.

TODO: add comments to code (explaining what each example is about)

- Integer
 - The key word is "int"
 - `var x = 6 // implicit var x : int = 6 // explicit`
- Strings
 - The key word is "string"
 - `var x = "abc" // implicit var x : string = "abc" // explicit`
- Booleans
 - The key word is "bool"
 - `var x = false // implicit var x : bool = false // explicit`
- Objects
 - The key word is "object"
 - `var x = 6 // implicit var x : object = 6 // explicit`
- Lists
 - The key word is "list"
 - Lists also have implicit vs explicit typing for the data-type it stores.
 - To set the explicit type, add angle brackets "<>" after the word list, that contain the data type.
 - Lists can also store lists.
 - `var x = [6, 4, 5] // implicit var x : list = [6, 4, 5] // explicit typing of an implicitly typed list`
`var x : list <int> = [6, 4, 5] // explicit typing of an explicitly typed list`
`var x : list <list> = [[6, 4], [2, 8], [1, 5]] // list of lists`

Assigning Variables

Once a variable is initialized, it can be assigned. Use the variable name, an equal sign "=", then the new value it will store.

```
var x = 1    // x is initialized
x = 2        // x is assigned to a new value (in this case 2)
```

Variable Scope

A variable's scope determines where a variable can be accessed, and how long it exists. * Global Variables * These variables are initialized outside of all loops, if statements, and function definitions. * Their lifespan is as long as the rest of the program. * Anywhere after initialization, the variable can be assigned and accessed in all of the program. * Local Variables * These variables are initialized inside of the body of either a loop, if statement, or function definition. * Their lifespan ends when the body it is initialized within ends. * After the variable's lifespan ends, the variable name is free to be used again.

Operations

Operations are a way to interact with other data, such as changing or comparing values.

Arithmetic Operations

Arithmetic operations are done on integer data-types, and return an integer.

When more than one arithmetic operation is in one expression, the operations are completed in PEMDAS order. First, operations with Parenthesis are done. Exponents are not supported in Catscript, so skip that one. Then Multiplication and Division are done. Finally, Addition and Subtraction. If there is more than one type of the same operation, those like items are evaluated in order from left to right. * Addition * Uses the "+" operator * `int x = 12 + 3 // x is set to 15` * Subtraction * Uses the "-" operator * `int x = 12 - 3 // x is set to 9` * Multiplication * Uses the "*" operator * `int x = 12 * 3 // x is set to 36` * Division * Uses the "/" operator * If the resulting number is not a whole number, the decimal is dropped. * `int x = 12 / 3 // x is set to 4`

```
int x = 10 / 3
// x is set to 3
```
```

## String Operations

- Concatination
  - Uses the "+" operator
  - Creates a new string that contains the string on the left hand side followed by the string on the right hand side
  - Note that string concatenation cannot be done between a string and any other data type. If the other type is not a string, then it is converted into a string.
  - ``` String x = "12" + "3" // x is set to 123

String x = "12" + 3 // x is set to 123

String x = "12" + null // x is set to 12null ```

## Comparisons

Comparisons compare the values of two different items. Comparison of greater than or less than can only be done between integers. Equality comparisons can be done with any type.

Order of evaluations: \* not (on booleans) \* less than or greater than \* equality \* not (if before parenthesis)

### Integer Comparisons

- less than
  - Uses the "<" operator
  - Evaluated to true if the left hand side number is less than the right hand side number.
  - 4 < 8 // true 4 < 4 // false 4 < 2 // false
- less than or equal to
  - Uses the "<=" operator
  - Evaluated to true if the left hand side number is less than or equal to the right hand side number. Otherwise, it evaluates to false.
  - 4 <= 8 // true 4 <= 4 // true 4 <= 2 // false
- less than or equal to
  - Uses the "<=" operator
  - Evaluated to true if the left hand side number is less than or equal to the right hand side number. Otherwise, it evaluates to false.
  - 4 >= 8 // false 4 >= 4 // true 4 >= 2 // true
- greater than
  - Uses the ">" operator
  - Evaluated to true if the left hand side number is greater than the right hand side number. Otherwise, it evaluates to false.
  - 4 > 8 // false 4 > 4 // false 4 > 2 // true

### Generalized Comparisons

The generalized comparisons of Catscript (equality) can be done between values of any type.

- Equality
  - Uses the "==" operator
  - Evaluated to true if the left hand side value is equal to the right hand side value. Otherwise, it evaluates to false.
  - 4 == 4 // true 4 == 8 // false "hi" == 4 // false null == null // true
- Not Equality
  - Uses the "!=" operator
  - Evaluated to true if the left hand side value is not equal to the right hand side value. Otherwise, it evaluates to false.

- `4 != 4 // false 4 != 8 // true "hi" != 4 // true null != null // false`

- Unary

- Not
- Uses the keyword "not" before a boolean value or expression to reverse it. True becomes false, false becomes true.
- If followed by anything other than true or false, it must have parentheses "()" surrounding the following expression.
- `not true // false not false // true not (5 == 6) // true`

## Features

---

### For loops

Catscript allows users to loop through the same chunk of embedded code with for loops. It might be more accurate to think of them as for each loops, and it iterates through every item in a set.

Start by using the keyword "for" followed by parenthesis "()". Within the parenthesis there is an identifier, then a separating comma, then the expression. After the parenthesis, there are curly braces "{}" and all the code within them is run once for each item in the expression. \* Think of *for (x in nums)*, the loop will iterate through the embedded code once for each item x in nums.

```
for (x in [1, 2, 4]) {
 print(x + " ")
}
// the printed output will be: 1 2 4
```

### Print statements

Users can display text in Catscript using print statements.

By using the keyword "print" followed by parenthesis "()", the expression within the parenthesis "()" will be printed.

```
print("Hello World")
// the printed output will be: Hello World
```

## Functions

A function allows a block of code be called elsewhere. Often functions perform a specific task, that can be called to run elsewhere in the program. This allows users to write reusable code, and increase program readability.

### Function Declaration

To declare a function, you start with the function name then can optionally follow with the function's return type. This is followed by parenthesis "()" that contain the parameters. Then there are curly braces "{}" that contain the body of the function. This is what will be executed when the function is run. Within the body you will need a return statement. \* Return details: Similarly to variables, functions can be implicitly or explicitly typed. Unlike variables, implicitly typed functions are set to type "void", which is an exclusive type for functions to say that there will be no returned value. \* Parameter details: There can be as many parameters as you like. Each parameter is entered in the format variable name optionally followed a colon ":" and its explicit data-type. If there is more than one, then the parameters are separated by a comma.

```

// Example of an implicitly typed function return that returns nothing (void)
printPlusHello(name : string) {
 print("Hello " + name + "!")
}

// This is an example of the format, but cannot run, since it doesn't have a valid type
//functionName : return-type () {
// // body
// // return (value) // return if the type is not void
//}

// Example of an explicitly typed function return that returns nothing (void)
printPlusHi : void (name : string) {
 print("Hi " + name + "!")
}

// Example of an explicitly typed function return that returns an int
convertCelsuisToFeriheit : int (Celsuis : int) {
 int ret = ...
 return (ret)
}

```

## Function Returns

Functions can return a single item. \* The type of the return is defined in the function declaration. \* When a return statement is reached, the function is exited. \* There must always be at least one return statement that is always reached. \* There can be multiple return statements in a function. \* No return statement is needed if the data-type is void. But one without a value can still be used to exit the function early

```
// All branches are covered with a return statement
functionName : string (x : bool) {
 if (x) {
 return ("x is true")
 } else {
 return ("x is false")
 }
}

// This will have a parser error, since there is no return if x is false.
functionName : string (x : bool) {
 if (x) {
 return ("x is true")
 }
}

// Function won't print hello because it returns before reaching the print statement
myfunc () {
 return ()
 print ("Hello")
}
```

## Function Calls

To access a function, you must call it. This can be done by typing the function name followed by parenthesis `()` containing all parameters that the function requires, separated by commas. If the function returns something, then it can be stored in a variable or used in an expression. The function call must have the same number of arguments as the function definition had parameters.

```
function functionName() {}

functionName() // function is called

function convertCelsuisToFeriheit : int (temp : int) {
 return (temp * 9 / 5 + 32)
}

var fahrenheit : int = convertCelsuisToFeriheit(0)
print(fahrenheit) // function is called, and the returned 32 is printed
```

## Conditional statements

Catscript allows users to make conditional statements using if statements. When the given expression evaluates to true, the embedded section of code is executed.

- If statements
  - In catscript if statements start with the keyword `"if"` followed by parenthesis `()`. Within the parenthesis `()` there is an expression that evaluates to a boolean. This is followed by curly braces `"{"` in which all code embedded within the curly braces is executed only if the initial expression evaluates to true.
  - `if (expression) { // all code embedded will be run only if the expression evaluated to true. print(x) }`

- Else if statements

- An else if statement must be put after either an if statement or another else if statement.
- This allows another expression to be checked only if the first expression evaluated to false.
- After the previous conditional's closing curly brace "}" the keywords "else if" must be used followed by a set of parenthesis "(" containing an expression that evaluates to a boolean. This is followed by curly braces "{}" in which all code embedded within the curly braces is executed only if the *else if* expression evaluates to true.
- `if (expression1) { print(x) } else if (expression2) { // all code embedded will be run only if expression1 evaluated to false and expression2 evaluated to true. print(x) }`

- Else statements

- An else statement must be put after either an if statement or another else if statement.
- After the previous conditional's closing curly brace "}" the keywords "else" must be used followed by curly braces "{}" in which all code embedded within the curly braces is executed only if the *else if* expression evaluates to true.
- `if (expression1) { print(x) } else if (expression2) { print(x) } else { // all code embedded will be run only if expression1 and expression2 evaluated to false. print(x) }`

`if (expression3) { print(x) } else { // all code embedded will be run only if expression3 evaluated to false. print(x) }`