CSCI 4020 Final Project

Mitch Nolte

Data Types - Primitives & Strings

- Primitives include:
  - Integers
  - Floating point numbers
  - Booleans
- Supported operations include:
  - Arithmetic operations between any numerical types
  - String concatenation for data of any type
    - Concatenating a variable with an empty string effectively casts it to a string
  - String repetition by multiplying it by an integer

```
val program = """
string1 = "Dog goes ";
string2 = "woof ";
println(string1 ++ string2 * 2);

pi = 3.1415;
radius = 5;
area = pi * radius*radius;
greaterThan20 = area > 20.0;

println(
    "Area of circle with radius "
    ++ radius ++ " is " ++ area
);

println("area > 20: " ++ greaterThan20);
"""

execute(program)
```

```
Dog goes woof woof
Area of circle with radius 5 is 78.537506
area > 20: true
```

Conditionals

- If, else, else if
- Equality tests for primitives and strings (must be the same type)
- Greater/less than comparison for numbers of the same type
- Logical and, or, not

```
val program = """
x = 5;
y = 6.0;
s = "string";
falseBool = false;

if(x > 5 && y > 5.0) {
    println("The numbers are large.");
} else if(x > 5 || y > 5.0) {
    println("One of the numbers is large");
} else {
    println("The numbers are small");
}

if(!falseBool) {
    if(s == "sTrInG") {
        println("The string is \"sTrInG\"");
    } else {
        println("The string is not \"sTrInG\"");
    }
}
"""

execute(program)
```

```
One of the numbers is large
The string is not "sTrInG"
```

Data Types - Collections

- Lists and dictionaries
- Elements can have any type, including other collections.
  - Dictionary keys must be a string or a primitive type however.
- Supports heterogeneous collections.
  - I.e. different types in the same collection.
- Elements can be accessed and modified with square bracket syntax.
  - Can also be used to insert new key/value pairs to a dictionary.

```
val program = """
// Lists
list = [5, 4, 3, 2, 1];
println(list);
println("List element 2 = " ++ list[2]);

list[3] = "string element";
list[2] = ["nested", "list"];
println(list);
println();

// Dictionaries
dict = { "key": "value", true: 5, false: 6, 100: "one hundred" };
println(dict);
println("Value associated with true = " ++ dict[true]);

dict[true] = "new value";
println(dict);
"""

execute(program)
```

```
[ 5, 4, 3, 2, 1 ]
List element 2 = 3
[ 5, 4, [ "nested", "list" ], "string element", 1 ]

{ 100: "one hundred", false: 6, "key": "value", true: 5 }
Value associated with true = 5
{ 100: "one hundred", false: 6, "key": "value", true: "new value" }
```

For Loops

- Can iterate over:
  - Integer range
  - List elements
  - Dictionary keys
    - Implemented as hash maps, so keys are not iterated over in any particular order.
- Integer ranges can be either ascending or descending.
  - Range is descending if the first number is greater than the second.

```
val program = """
list = [45, 62, 87];
dict = {"key 1": 54.45, "key 2": 26.62, "key 3": 78.87};

listSum = 0;
for(element in list) {
    listSum = listSum + element;
}
println("List sum = " ++ listSum);

dictSum = 0;
for(key in dict) {
    dictSum = dictSum + dict[key];
}
println("Dictionary sum = " ++ dictSum);
println();

for(i in 1..5) {
    for(j in 5..i) {
        print(j);
    }
    println();
}
"""

execute(program)
```

```
List sum = 194
Dictionary sum = 159.94

54321
5432
543
54
5
```

Functions

Supports nested functions and recursion.

```
val program = """
function greeting(name, message) {
    function introduction(firstName) {
        "Hi, my name is " ++ firstName;
    }

    println(introduction(name));
    println(message);
}

greeting("Albert", "How are you?");
"""

execute(program)
```

```
Hi, my name is Albert
How are you?
```

```
val program = """
function factorial(n) {
    if(n < 2) {
        1;
    } else {
        n * factorial(n-1);
    }
}

println(factorial(10));
"""

execute(program)
```

```
3628800
```