Loops

2-24-2012

Opening Discussion

- Minute essay comments:
  - What books would I read if I had time?
  - Impact of the power outage on our schedule. (IcPs)
  - Cheat sheet can have front and back.
  - Special characters in encryption/decryption.
  - Looking at code: Google Code and Github.
  - All scripts ARE online.
  - Significant functions: map and filter.
  - How to practice and study.
  - List of List?

More

- Reading BigInts.
- Sorting – that is later in the semester.
- Coding augmented reality and CV.
- Uses of HUD.
- “Books website”
- Breaking out in dance.
- 5 IcPs
- Is stumbling along “normal”?
- Games in majors lab.
- The unzip method.

Variable Length Argument Lists

- You can make functions that don't specify exactly how many arguments they take.
- These are often called var-args.
- To do this, but a * after the type. It can only be the last argument in a list.

Calling Var-Args with Collections

- It is often helpful to call a var-args method passing a collection for the variable length arguments.
- You can do this, but you have to tell Scala what you are doing.
- Follow the collection with :_* to do this.
- The : is like specifying a type.
- The _ says you don't care about the exact type.
- The * is like the * in var-args declarations.

Aliasing and Mutability

- I argue that immutable collections like Lists can be safer than mutable ones like Arrays.
- One of the big reasons for this is aliasing.
- An alias in programming is just like in normal life. It is a second name for something.
- Variables are really references to objects.
- If a second variable is assigned the same value as the first, they are aliases to that object.
- Let's play with this and draw on the board.

Aliasing for Argument Passing

- When you pass arguments, you are really passing references.
- So arguments in functions are aliases to the objects outside the function
- If the object is mutable, the function can change it.

Pass-by-Name

- There is another way to pass things in Scala called pass-by-name.
- When you pass something by name, it isn't evaluated at the time it is passed. Instead it is turned into a function and that function is evaluated every time the variable is used.
- The syntax is to put an => before a type, but not have an argument list before the arrow.

Fill and Tabulate

- There are two other ways of creating collections: fill and tabulate. Both are curried. Second argument to fill is by name, second argument to tabulate is a function.
- The fill method on Array or List takes a first argument of how many elements. After that is a by-name parameter that gives back the type you want in the array or list.
- Tabulate also takes a size first. After that is a function that takes the index.

while Loop

- Recursion is sufficient for making repetition, but in imperative languages it isn't the normal approach. Instead, people use loops.
- The simplest loop is the while loop.
  - while(*condition*) *statement*
- The condition is evaluated first. If it is true the statement (possibly a block) executes.
- This repeats until the condition is false.

do-while Loop

- The partner to the while loop is the do-while loop.
  - do {
    - *statement*
  - } while(*condition*)
- This loop is post-check instead of the pre-check of the normal while loop.
- Always happens once.
- The while loop might never happen.

The for Loop

- The most commonly used loop in most languages is the for loop. The Scala version is a bit different from most.
- Often used for counting:
  - for(i <- 1 to 10) { ... }
- In general it is a “for each” loop that goes through a collection.
  - for(*e* <- *coll*) { ... }
- Variable takes on value of each element in the collection.

Range Type

- Range types provide an easy way to make collections for counting.
- “to” and “until” operate on numeric types to produce ranges.
  - 1 to 10
  - 0 until 10
- Use “by” to change the stepping in a range.
  - 1 to 100 by 2
  - 10 to 1 by -1
  - 'a' to 'z' by 3

yield

- The for loop can be used as an expression if you put yield between the end of the for and the expression after it.
  - for(*e* <- *coll*) yield *expr*
- What you get back will be a collection that is generally of the same type as what you iterated over.

if Guards

- You can put conditions in the for that will cause some values to be skipped.
  - for(n <- nums; if(n%2==0)) ...

Multiple Generators

- You can also put multiple generators in a for loop.
  - for(i <- 1 to 10; j <- i to 10) ...
- You can combine as many generators and guards as you want. You can also declare variables in the middle of the for.
- The thing you assign into is like a val so it can be a “pattern”. We have only seen this with tuples so far.

Multidimensional Arrays

- You can have collections of collections. A common example would be something like Array[Array[Double]] to represent a matrix.
- Both fill and tabulate can be used to make these.
  - val ident=Array.tabulate(3,3)((i,j) => if(i==j) 1.0 else 0.0)

Minute Essay

- Any questions?
- Midterm is on Monday. What times over the weekend work well for you to have a review session? I will also do a Google+ hangout review session during the weekend.