CSE231 - Lab 02

Comparison Operators, Boolean Logic, Conditionals, Loops

Is anyone still confused about the format of this class?

Comparison Operators

We’ve talked about mathematical operators, your classic addition, subtraction, multiplication, etc. But in all programming languages, there are also comparison operators that return a boolean.

- >, greater than
- <, less than
- >=, greater than or equal to
- <=, less than or equal to
- ==, equal to
- !=, not equal to

All of them require a left-hand value and a right-hand value.

Conditional Statements

You usually use boolean operations in combination with a conditional statement, and these conditional statements are “linked” when using them together.

- if, if my condition is True, we run this code
- elif, else if *this* condition is True (and all previous conditions were False), we run *this* code
- else, if none of the other conditions were True, we run *this* code

not

In addition to the comparison operators, there are 3 comparison keywords as well, but I’ll only be showing you one of them for now.

The ‘not’ keyword results in True if the statement evaluates to not True (False)

```
-   not True == False    # True
-   not False == True    # True
-   5 > 4        # True
-   not 5 > 4    # False
```

You can think of it as just taking the inverse of whatever boolean statement you’re evaluating.

While-loop

A while-loop continuously checks a condition until it is False. Like if-statements, it requires a colon and an indented block of code.

Important to note, is that the condition is checked when first evaluated from proceeding lines, and then gets reevaluated after running through its respective block of code for each iteration.

There are some programming languages that have variations on this concept.

L2-2

For-loop / range()

A for-loop iterates for a specified amount of times or over a collection as opposed to checking a condition like the while-loop. Commonly, a for-loop used to iterate for a certain range of numbers is written as ‘for i in range(n):’, where ‘i’ is an iterative integer that starts from 0, and ends at ‘n-1’.

range([start], stop, [step])

range() is a strange function. When 1 parameter is given, that becomes the ‘stop’ parameter. When 2 are given, the first becomes the ‘start’, the second becomes the ‘stop’. You can then add a third, ‘step’ when you have the first 2 given.

The three parameters do about what you would expect.

L2-3

break

Alternatively, if you don’t want to iterate through your loop again, you can preemptively interrupt it using the aptly named ‘break’ statement.

Typically, you would be checking a certain, outside condition where you wouldn’t want your loop to continue.

‘break’ is a keyword that stands alone in-line.

continue

The opposite of the break-statement would likely be this, the continue-statement. Instead of preemptively breaking out of the loop, ‘continue’ preemptively repeats the loop.

Like ‘break’, you would typically use ‘continue’ in special cases for your loop. It again sits alone in-line.

What’s so special about break/continue?

break and continue, importantly, skip all lines below themselves that are within the same loop-block. They can *only* be used inside loop structures.

This, combined with if/elif/else statements, make it so that *the order in which you put your statements matter*. You can take advantage of this, but it can also be a hindrance at times. Be careful with this stuff.

This is really important, and you should keep it in mind because it will come up frequently.

L2-4

👏Lab👏Time👏

Hopefully you guys have caught on to the pattern, now. Go to the CSE course website/repo and find the lab for today (Lab 02).

github.com/braedynl/CSE231-020-SS20 / www.cse.msu.edu/~cse231/

Follow the PDF instructions and have at it. Remember that you can always ask me about Mimir homework and projects as well.

Show me your finished lab with the test cases passed and you’re free to go.

Call me over if you have questions, comments, concerns or just want to talk.