Blocks

- What is syntax (delimiters)
- Where can blocks be used
- Scope and blocks
- Do blocks return a value (use in expressions)
- Entry point(s) to blocks
- Exit point(s) from blocks

Block syntax and use

- Typically need start/stop delimiters, e.g. { }, begin end, indentation level, etc. Again, one of the most recognizable aspects of language syntax
- Can blocks be used anywhere you could use a single statement, or are blocks implicitly used as parts of more complex control structures (loops, if statements, etc)?
- Do blocks have a return value, i.e. can you use them in an expression, and how do they return a value (syntax)?

Block scope

- Does each block have its own scope?
- If so, what are the nesting rules (e.g. lexical or dynamic, how are overlapping scope refereces handled)?
- Are the rules consistent across all types of block (e.g. the same in if statements as loops, and as in switch statements, etc)?

Exit point(s)

- At what points can you leave a block? E.g. return statements, break statements, after executing last statement in block, etc)
- Is there a last/final mechanism to specify actions that are always applied when leaving the block, regardless of which exit point was used?

Entry points

- Most loops have a unique first instruction that is executed upon entering loop (typically the “top” instruction in the loop)
- Is it possible to have multiple entry points to a loop, a variety of places you could jump in to and start at? If so, what happens if some of those entry points “skip” instructions like variable declarations/initializations (see C switch example)?