In [1]:
```
# Styling notebook
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]:

Scanner Example

It's a sketch for a Scanner adapted to a "normal" C/Java-type imperative language. You might need to rewrite some parts depending on the token structure in your source language - YMMV.

Most of the "theory" is in the short lecture, nothing really interesting to see here. But remember one thing:

The **Ironclad Scanner Rule (ISR)**: *The Scanner positions itself ALWAYS one character behind the the current lexeme.*

In [2]:
```
import string

class Scanner :

    # Tons of character classes, tables etc
    EOI = '$'
    START_COMMENT = '#'
    END_COMMENT = '#'
    START_STRING = '"'
    END_STRING = '"'
    EQUAL = '='
    NOT = '!'
    GREATER = '>'
    LESS = '<'

    WHITESPACE = {' ', '\t', '\n'}
    DIGITS = {'0','1','2','3','4','5','6','7','8','9'}
    LETTERS = set(string.ascii_uppercase).union(set(string.ascii_lowercase))
    LETTERS_OR_DIGITS = LETTERS.union(DIGITS)

    OP_TABLE = {
        '(' : 'lParen',
        ')' : 'rParen',
        '{' : 'lCurly',
        '}' : 'rCurly',
        '+' : 'plusSym',
        '-' : 'minusSym',
        '*' : 'timesSym',
        '/' : 'divSym',
        ';' : 'semicolon',
        ',' : 'comma',
    }

    KEYWORD_TABLE = {
        'while'  : 'whileSym',
```

```
        'return' : 'returnSym',
        'if'     : 'ifSym',
        'else'   : 'elseSym',
        'do'     : 'doSym',
        'int'    : 'intSym',
        'string' : 'stringSym',
    }

    eoIToken = 'eoI'

    def __init__(self, source) :
        self.source = source + Scanner.EOI

        # Initialize Scanner state
        self.init()

    # A. Lowest level functions
    # Most of these need to be rewritten to fit into your source format
    # Initialize Scanner state
    def init(self) :
        self.position = 0           # First character
        self.currentText = None     # Current text/token
        self.currentToken = None    # Nothing beats good naming, right?

    # Error messages
    def error(self, message) :
        print(">>> Error: ", message)

    # Abstraction from current character from the input
    def currentCh(self) :
        return self.source[self.position]

    # Abstraction from reading the next character
    def move(self) :
        # Have to do error check
        self.position += 1

    # Abstraction from having read the last character
    # Here it's the special EOI character in the input STRING!
    def atEOI(self) :
        return self.currentCh() == Scanner.EOI

    # B. Second level movements

    # Same as move, but report error of moving past EOI
    def eat(self) :
        if self.atEOI() :
            self.error('Cannot move beyond EOI!')
        else :
            self.move()

    # Find a character "x" down the input stream
    # Report error if none found before EOI
    # Collect all characters found in a result string
    def find(self,x) :
        result = ''
        while self.currentCh() != x and  not self.atEOI()  :
            result = result + self.currentCh()
            self.eat()
        if self.atEOI() :
```

```
            self.error('EOI detected searching for '+ x)
        else :
            return result

    # Same as find, but look for any character in the set "s"
    def findStar(self,s) :
        result = ''
        while self.currentCh() not in s and not self.atEOI() :
            result = result + self.currentCh()
            self.eat()
        if self.atEOI() :
            self.error('EOI detected searching for '+ s)
        else :
            return result

    # Run over character "x" down the input stream
    # Report error if none other found before EOI
    # Collect all characters found in a result string
    def skip(self,x) :
        result = ''
        while self.currentCh() == x :
            result = result + self.currentCh()
            self.eat()
        return result

    # Same as skip but for a set "s"
    def skipStar(self,s) :
        result = ''
        while self.currentCh() in s :
            result = result + self.currentCh()
            self.eat()
        return result

    # Skip over all whitespaces
    def skipWS(self) :
        self.skipStar(Scanner.WHITESPACE)

    # Skip over everything up to end of comment
    # Assuming you're still standing on begin of comment
    def skipComment(self) :
        self.eat()    # Move forward --> first character of comment
        self.find(Scanner.END_COMMENT)
        self.eat()    # Move past end of comment

    # Move over whitespaces or comment (based on current character)
    def jump(self) :
        if self.currentCh() in Scanner.WHITESPACE :
            self.skipWS()
        elif self.currentCh() == Scanner.START_COMMENT :
            self.skipComment()

    def jumpStar(self) :
        while self.currentCh() in Scanner.WHITESPACE or self.currentCh() == Scanner.ST
            self.jump()

    # C. High level tokenizers

    # Skip over digits : return integer value
    # Don't eat, will lose first character
    def NUM(self) :
```

```
        return 'numConstant', int(self.skipStar(Scanner.DIGITS))

    # Skip over letters/digits : return string scanned
    # Don't eat, will lose first character
    def ID(self) :
        return 'identifier', self.skipStar(Scanner.LETTERS_OR_DIGITS)

    # Skip over everything up to end of string : return string scanned
    # Now you have to eat, else you get the begin of string character back
    def STRI(self) :
        self.eat()
        chars = self.find(Scanner.END_STRING)
        self.eat()
        return 'stringConstant', chars

    # Scan a possibly two-character token, like == or >=
    # Yes, it's generally the = that comes in the second place
    # Return firstToken, if there's something other than secondCh is next
    # Return secondToken, if secondCh is next
    # Main topic here: position yourself correctly in any case
    def twoCharSym(self,secondCh,firstToken,secondToken) :
        self.eat()
        if self.currentCh() == secondCh :
            self.eat()
            return secondToken
        else :
            return firstToken

    # C. Main tokenizer

    # Return class and lexeme of the next token
    # 'eoI' is artificial and denotes "End of Input"
    # 'Unknown' is a class for a lexeme that doesn't translate into a proper token
    # Lots of assumptions in this, needs to be rewritten if
    #    a) we have floating-point constants
    #    b) we have alphanumerical variable names
    #    c) it's Monday

    def nextToken(self) :
        # Trivial test of EOI (End Of Input)
        if self.atEOI() :
            return Scanner.eoIToken, None # Yep! Out of here

        # Find next token start
        # Note that this doesn't move if a "good"" character is current
        self.jumpStar()

        # Ok, now let's check the character we're on
        c = self.currentCh()

        # All the possibilities (hopefully)

        # Arbitrary long tokens (numbers, strings, ids/reserved words)
        if c in Scanner.DIGITS : return self.NUM()
        if c == Scanner.START_STRING : return self.STRI()
        if c in Scanner.LETTERS :
            token, string = self.ID()
            # Isolated the string: Is it a reserved word?
            if Scanner.KEYWORD_TABLE.get(string,None) != None :
                return Scanner.KEYWORD_TABLE[string], None      # Yes, return the toke
```

```
            else :
                return token, string                            # No, just an id: retu

        # Two-char tokens: ==, !=, >=, <=
        if c == Scanner.EQUAL : return self.twoCharSym(Scanner.EQUAL,'assignSym','equa
        if c == Scanner.NOT : return self.twoCharSym(Scanner.EQUAL,'notSym','notEqualS
        if c == Scanner.GREATER : return self.twoCharSym(Scanner.EQUAL,'greaterSym','g
        if c == Scanner.LESS : return self.twoCharSym(Scanner.EQUAL,'lessSym','lessEQS

        # One-char tokens: Check in Op table
        if Scanner.OP_TABLE.get(c,None) != None :
            self.eat()
            return Scanner.OP_TABLE[c], None

        # Shrug it off, no idea!
        return None, None
```

In [3]:

```
# s = Scanner('int xyz = 12; # Now the while loop # while (xyz >= 0) { xyz = xyz - 1;}

s = Scanner('int xyz,int a = 12,13; # Now the while loop # while (xyz >= 0) { xyz = xy

tok, text = s.nextToken()
while tok != Scanner.eoIToken :
    print(tok,': ',text)
    tok, text = s.nextToken()
```

```
intSym :  None
identifier :  xyz
comma :  None
intSym :  None
identifier :  a
assignSym :  None
numConstant :  12
comma :  None
numConstant :  13
semicolon :  None
whileSym :  None
lParen :  None
identifier :  xyz
greaterEQSym :  None
numConstant :  0
rParen :  None
lCurly :  None
identifier :  xyz
assignSym :  None
identifier :  xyz
minusSym :  None
numConstant :  1
semicolon :  None
rCurly :  None
```