NOTE: This ~BNF grammar was what we drafted up initially to help with the spec. The canonical grammar is now in the code.

-- Top level description of whole program
-- if normal
Program := Header Docs UseList GroupList FilterList ‘{‘ ComputationList ‘}’

-- if .grp file
Program := GroupList

-- Script header, ex: script foo, script foo(x, y). This section is mandatory.
-- x and y are in top level scope. Don’t know type yet.

Header:= “script” fileName ‘\n’
| “script” fileName Args ‘\n’

Args := ‘(‘ Args_List ‘)’

Args_List := Var
| Var ‘, ’ Args_List

-- Documentation. Section is mandatory. Calling with a --help compiler flag will return this string.
-- Should not be in AST (doesn’t need to be typechecked). But should be in parser anyway to check if it is not there.

Docs := ‘/**’ StrLit ‘*/’

-- Use clauses, ex: use foo.grp, bar.grp
-- All names must end in .grp
-- Files must exist, else semantic error
-- All of them in single line (might not be readable, consider changing?)

UseList := ‘use’ UseStmt ‘\n’ |

UseStmt := fName | fName ‘, ’ UseStmt

fName := StrLit ‘.grp’

-- Group definitions, ex: group x = {“red”, “blue”} or group Id ids = {“1”, “2”}
-- Can only support strings (and they have to be quoted!)

GroupList := GroupDef | GroupDef ‘\n’ GroupList |

GroupDef := ‘group ’ GroupType Var ‘=’ ‘{ ‘ StrGrpList ‘}’

GroupType := Id | Sex | Birthyear | Diagnosis | Days | Years | Postalcode | Date | Hour | Event |

StrGrpList := StrGrp | StrGrp ‘, ‘ StrGrpList

StrGrp := ‘“‘ StrLit ‘“‘ | ‘<’ StrLit ‘>’

-- Filters. ex: population is
Id: <patientGroupOne>
Birthyear: <patientBirthyearRange>
Sex: M, F

FilterList := Filter | Filter FilterList |

Filter := DoctorTag Verb ‘\n’ DocFieldList | PopulationTag Verb ‘\n’ PopFieldList | PeriodTag Verb ‘\n’ PeriodFieldList | EventTag Verb ‘\n’ EventQual

-- proof of concept for adding different languages
DoctorTag := ‘doctor’ | ‘doctors’ | ‘medicine man’ | ‘medicine woman’ | ‘medicine person’ | ‘witch (who hasn’t been burnt at stake)’ | ‘witch who has been burnt at the stake’ | ‘shaman’ | ‘soothsayer’ | ‘healer’ | ‘cleric’ | ‘priest’ | ‘MD’ | ‘Dr. House’

PopulationTag := ‘populations’ | ‘population’

PeriodTag := ‘period’ | ‘periods’

EventTag := ‘event’ | ‘events’

Verb := ‘are’ | ‘is’ | ‘be’ | ‘was’| ’will be’ | ‘being’ |

DocFieldList := DocField | DocField ‘\n’ DocFieldList |

-- Optional. ex: doctor is
id: 10, 12 to 14

oncologist: yes

DocField := ‘id : ’ NumRangeList | ‘oncologist : ’ Boolean

NumRangeList := NumRange | NumRange ‘, ’ NumRangeList |

NumRange := Num | Num ‘ to ’ Num

PopFieldList := PopField | PopField ‘\n’ PopFieldList |

PopField := ‘id : ’ YearList | ‘gender : ’ GenderList | ‘birthyear : ’ YearList | ‘diagnosis : ’ DiagnosisList | ‘postalcode : ’ PostalcodeList

YearList := Year | Year ‘, ’ YearList |
… so on for all fields

-- define Year and Postalcode in lexer. Also Month.

Gender := ‘Male’ | ‘M’ | ‘Female’ | ‘F’ | ‘Non binary’ | ‘NB’

PeriodFieldList := ‘years : ’ Year | ‘months : ’ Months | ‘days : ’ Days | ‘hours : ’ Hours | ‘start : ’ ??? | ‘end : ’ ???

EventFieldList := EventField | EventField ‘\n’ EventFieldList |

EventField := **-- events list, slide 30 of oncotime slides**

-- how to handle groups in fields?

-- Computations. Mandatory.

ComputationList := Computation | Computation ‘\n’ ComputationList

Computation := Foreach | Table | Sequence | Print | Table | Barchart

Foreach := ‘foreach ’ CompType Var | ‘foreach element’ Var ‘ of ’ Var

CompType := ‘diagnosis’ | ‘patient’ | ‘doctor’

Table := ‘table ’ Var ‘ = ’ ‘count ’ CompCount

CompCount := ‘patient by’ PatientField | ‘doctor by’ DoctorField | ‘diagnosis by’ DiagField

PatientField := ID | Gender | Birthyear | Diagnosis | Postalcode

DoctorField := ID | Oncologist

DiagField := Name

Barchart := ‘barchart ’ Var

Print := ‘print ’ Expr

Expr := Var | Var.length | Var ‘[‘ Var ‘]’

Sequence := ‘list ’ Var ‘ = ’ ‘sequences like ’ EventList

EventList := EventField ‘ -> ’ EventList | EventField
