Abstract Syntax Tree

	Abstract Classes	Concrete classes
	ASTNode	
<Program> ::= <ImportList> class IDENT <Block>		Program
<ImportList> ::=(import IDENT (. IDENT)* ;) *		List<QualifiedName> in enclosing Program
	BlockElem	
<Block> ::= { (<Declaration> ; | <Statement> ;)* }		Block Block contains a list of BlockElem
	Declaration extends BlockElem	
<Declaration> ::= def <VarDec>		VarDec extends Declaration
<Declaration>::= def <ClosureDec>		ClosureDec extends Declaration
<VarDec> ::= IDENT (: <Type> | ϵ)		
	Type	
<Type> ::= <SimpleType>		SimpleType extends Type
<Type> ::= <KeyValueType>		KeyValueType extends Type
<Type> ::= <ListType>		ListType extends Type
<SimpleType> ::= int | boolean | string		
<KeyValueType> ::= @@ [<SimpleType> : <Type>]		KeyValueType
<ListType> ::= @ [<Type>]		ListType
<ClosureDec> ::= IDENT = <Closure>		ClosureDec extends Declaration
<Closure> ::= { <FormalArgList> -> (<Statement> ;)*}		Closure
<FormalArgList> ::= ϵ | <VarDec> (, <VarDec>)*		List<VarDec> in Closure

	Statement	
<Statement> ::= <LValue> = <Expression>		AssignmentStatement extends Statement
<Statement> ::= print <Expression>		PrintStatement extends Statement
<Statement> ::= while (<Expression>) <Block>		WhileStatement extends Statement
<Statement> ::= while* (<Expression>) <Block>		WhileStarStatement extends Statement
<Statement> ::= while* (<RangeExpression>) < Block>		WhileRangeStatement extends Statement
<Statement> ::= if (<Expression>) <Block>		IfStatement extends Statement
<Statement> ::= if (<Expression>) <Block> else <Block>		IfElseStatement extends Statement
<Statement> ::= %<Expression>		Expression Statement
<Statement> ::= return <Expression>		Return Statement
<Statement> ::= ϵ		An empty statement should not add anything to the AST
	Expression	
<ClosureEvalExpression> ::= IDENT (<ExpressionList>)		ClosureEvalExpression
	LValue	
<LValue> ::= IDENT		IdentLValue
<LValue> ::= IDENT [<Expression>]		ExpressionLValue
<List> ::=@ [<ExpressionList>]		ListExpression
<ExpressionList> ::= ϵ | <Expression> (, <Expression>)*		List<Expression> in enclosing ListExpression
<KeyValueExpression> ::= <Expression> : <Expression>		KeyValueExpression
<KeyValueList> ::= ϵ | <KeyValueExpression> (, <KeyValueExpression>) *		List<KeyValueExpression> in enclosing MapListExpression
<MapList> ::= @@[<KeyValueList>]		MapListExpression
<RangeExpr> :: <Expression> .. <Expression>		RangeExpression

<Expression> ::= <Term> (<RelOp> <Term>)*		BinaryExpression or result of Term
<Term> ::= <Elem> (<WeakOp> <Elem>)*		BinaryExpression or result of Elem
<Elem> ::= <Thing> (<StrongOp> <Thing>)*		BinaryExpression or result of Thing
<Thing> ::= <Factor> (<VeryStrongOp> <Factor)*		BinaryExpression or result of Factor
<Factor>::= IDENT		IdentExpression
<Factor>::=IDENT [<Expression>]		ListOrMapElemExpression
<Factor>::= INT_LIT		IntLitExpression
<Factor ::= true		BooleanLitExpression
<Factor>::= false		BooleanLitExpression
<Factor>::= (<Expression>)		result of <Expression>
<Factor>::= ! <Factor>		UnaryExpression
<Factor>::= -<Factor>		UnaryExpression
<Factor>::= size(<Expression>)		SizeExpression
<Factor>::= key(<Expression)		KeyExpression
<Factor>::= value(<Expression >)		ValueExpression
<Factor>::= <ClosureEvalExpression>		ClosureEvalExpression
<Factor>::= <Closure>		ClosureExpression
<Factor>::= <List>		ListExpression
<Factor>::= <MapList>		MapListExpression
	Operators are fields in enclosing BinaryExpression or UnaryExpression	
<RelOp> ::= | | & | == | != | < | > | ≤ | ≥		
<WeakOp> ::= + | -		
<StrongOp> ::= * | /		
<VeryStrongOp> ::= ≪ | ≫		