

Delimiters

$\langle singleQuote \rangle ::= ' '$

$\langle doubleQuote \rangle ::= ''$

$\langle terminator \rangle ::= ';' \mid '\backslash n'$

Primitive Types

$\langle int \rangle ::= [\text{integer}]$

$\langle bool \rangle ::= \text{'true'} \mid \text{'false'}$

$\langle char \rangle ::= \langle singleQuote \rangle [\text{character}] \langle singleQuote \rangle$

$\langle string \rangle ::= \langle doubleQuote \rangle [\text{character}^*] \langle doubleQuote \rangle$

$\langle null \rangle ::= \text{'null'}$

Algebraic Data Types and Type-Traits

$\langle generic \rangle ::= '[\langle ident \rangle [', \langle ident \rangle]^*]'$

$\langle adt \rangle ::= \text{'type'} \langle ident \rangle [\langle generic \rangle] [': ' \langle ident \rangle [\langle generic \rangle]] \{ '[\langle ident \rangle ': ' \langle type \rangle [', \langle ident \rangle ': ' \langle type \rangle]^*] ' \}$

$\langle typeclass \rangle ::= \text{'typeclass'} \langle ident \rangle \{ '[\langle ident \rangle = ' \langle type \rangle [', \langle ident \rangle = ' \langle type \rangle]^*] ' \}$

$\langle instance \rangle ::= \text{'instance'} \langle ident \rangle ': ' \langle ident \rangle \{ ' \langle prog \rangle ' \}$

Types

$\langle type \rangle ::= \text{'int'} \mid \text{'bool'} \mid \text{'char'} \mid \text{'string'} \mid \text{'null'}$
| $\langle type \rangle \text{'->'} \langle type \rangle$
| $\text{'('} [\langle type \rangle [', ' \langle type \rangle]^*] \text{')'}$ $\text{'->'} \text{'('} [\langle type \rangle [', ' \langle type \rangle]^*] \text{')'}$
| $\text{'List'} \mid \text{'Array'} \mid \text{'Set'} \text{'('} [\langle type \rangle]$
| $\text{'Tuple'} \text{'('} [\langle type \rangle [', ' \langle type \rangle]^*]$
| $\text{'Dict'} \text{'('} [\langle type \rangle ', ' \langle type \rangle]$
| $\langle ident \rangle [\text{'('} [\langle type \rangle]$

Arithmetic and Boolean Operators

$\langle arithOp \rangle ::= '+' \mid '-' \mid '*' \mid '/' \mid \text{'%'}$

$\langle boolOp \rangle ::= '<' \mid '>' \mid '<=' \mid '>=' \mid '!' \mid '!=' \mid '==' \mid '\&\&' \mid '||'$

$\langle op \rangle ::= \langle arithOp \rangle \mid \langle boolOp \rangle$

Functions

$\langle arg \rangle ::= \langle ident \rangle ' : ' \langle type \rangle ['=' \langle atom \rangle]$

$\langle lowerBoundOp \rangle ::= '>:'$

$\langle upperBoundOp \rangle ::= '<:'$

$\langle bounding \rangle ::= [\langle lowerBoundOp \rangle \langle ident \rangle] [\langle upperBoundOp \rangle \langle ident \rangle]$

$\langle templateTypes \rangle ::= ' [' \langle ident \rangle \langle bounding \rangle [' , ' \langle ident \rangle \langle bounding \rangle]^* ']'$

$\langle funDef \rangle ::= ' \mathbf{fn} ' \langle ident \rangle [\langle templateTypes \rangle] ' (' [\langle arg \rangle [' , ' \langle arg \rangle]^*] ') ' ['->' \langle type \rangle] '=' \langle smp \rangle \langle terminator \rangle$

$\langle prog \rangle ::= [\langle funDef \rangle]^*$

$\langle app \rangle ::= \langle atom \rangle [[' [' \langle type \rangle [' , ' \langle type \rangle]^* '] ']^* [' (' [\langle smp \rangle [' , ' \langle smp \rangle]^* ') ']^*]$

$\langle anonLmbd \rangle ::= ' \backslash ' \langle arg \rangle ['->' \langle type \rangle] '=' \langle smp \rangle$

Pattern Matching and Switches

$\langle caseVal \rangle ::= \langle type \rangle | \langle smp \rangle$

$\langle match \rangle ::= ' \mathbf{match} ' ' (' \langle smp \rangle ') ' ' \{ ' \mathbf{case} ' \langle caseVal \rangle '=' \langle smp \rangle [' \mathbf{case} ' \langle caseVal \rangle '=' \langle smp \rangle]^* ' \}$

$\langle matchSwitch \rangle ::= \langle match \rangle | \langle switch \rangle$

Expressions

$\langle atom \rangle ::= \langle int \rangle | \langle bool \rangle | \langle char \rangle | \langle string \rangle | \langle null \rangle$
| $' (' \langle smp \rangle ') '$
| $\langle ident \rangle [' . ' \langle ident \rangle]^*$

$\langle tight \rangle ::= \langle app \rangle [' | > ' \langle app \rangle]$
| $' \{ ' \langle exp \rangle ' \}$

$\langle utight \rangle ::= [\langle op \rangle] \langle tight \rangle$

$\langle smp \rangle ::= \langle utight \rangle [\langle op \rangle \langle utight \rangle]$
| $' \mathbf{if} ' ' (' \langle smp \rangle ') ' \langle smp \rangle [' \mathbf{else} ' \langle smp \rangle]$
| $' \mathbf{List} ' | ' \mathbf{Tuple} ' | ' \mathbf{Array} ' | ' \mathbf{Set} ' ' \{ ' [\langle smp \rangle [' , ' \langle smp \rangle]^*] ' \}$
| $' \mathbf{Dict} ' ' \{ ' [\langle smp \rangle ' : ' \langle smp \rangle [' , ' \langle smp \rangle ' : ' \langle smp \rangle]^*] ' \}$
| $\langle matchSwitch \rangle$
| $\langle typeclass \rangle$
| $\langle instance \rangle$
| $\langle adt \rangle$

```

|   $\langle prog \rangle$ 
|   $\langle anonLmbd \rangle$ 

 $\langle exp \rangle ::= \langle smp \rangle [\langle terminator \rangle \langle exp \rangle]$ 
|   $[\text{‘lazy’}] \text{‘val’} \langle ident \rangle [\text{‘:’} \langle type \rangle] \text{‘=’} \langle smp \rangle \langle terminator \rangle \langle exp \rangle$ 
|   $\text{‘include’} \langle file \rangle \langle terminator \rangle \langle exp \rangle$ 

```