

MiniPython – A Small Subset of Python Language

1 INTRODUCTION

MiniPython is a toy language for the [COSE212](#) course at Korea University. MiniPython is inspired by the Python programming language.¹ The name MiniPython indicates that it is a minimal subset of the Python language, and it supports the following features:

- **pass statements** (`pass`)
- **variable assignment statement** (`x = e`)
- **set item** (`x[e] = e`)
- **if-statements** (`if-elif-else`)
- **while-statements** (`while`)
- **for-statements** (`for-in`)
- **loop control statements** (`break` and `continue`)
- **try and raise statements** (`try-except` and `raise`)
- **function definitions** (`def f(...):`)
- **return statements** (`return`)
- **yield and yield from statements** (`yield` and `yield from`)
- **none value** (`None`)
- **number (integer) values** (`0, 1, -1, 2, -2, 3, -3, ...`)
- **boolean values** (`True` and `False`)
- **arithmetic operators**: negation (`-`), addition (`+`), subtraction (`-`), multiplication (`*`), division (`//`), and modulo (`%`)
- **logical operators**: conjunction (`and`), disjunction (`or`), and negation (`not`)
- **comparison operators**: equality (`==`, `!=`, `is`, and `is not`) and relational (`<`, `>`, `<=`, and `>=`)
- **lists** (`[e, ..., e]`)
- **list append method** (`e.append(e)`)
- **get item** (`x[e]`)
- **lambda functions** (`lambda ...: e`)
- **function applications** (`e(...)`)
- **conditional expressions** (`e if e else e`)
- **iter function** (`iter(e)`)
- **next function** (`next(e)`)

This document is the specification of MiniPython. First, Section 2 describes the concrete syntax, and Section 3 describes the abstract syntax with the desugaring rules. Then, Section 4 describes the small-step operational (reduction) semantics of MiniPython.

2 CONCRETE SYNTAX

The concrete syntax of MiniPython is written in a variant of the extended Backus–Naur form (EBNF). The notation `<nt>` denotes a nonterminal, and "t" denotes a terminal. We use `?` to denote an optional element and `+` (or `*`) to denote one or more (or zero or more) repetitions of the preceding element. We use `butnot` to denote a set difference to exclude some strings from a producible set of strings. We omit some obvious terminals using the ellipsis (`...`) notation. The `<INDENT>` and `<DEDENT>` denote the increase and decrease of the indentation level, respectively.

¹<https://www.python.org/>

```

// basic elements
<digit> ::= "0" | "1" | "2" | ... | "9"
<number> ::= "-"? <digit>+
<alphabet> ::= "A" | "B" | "C" | ... | "Z" | "a" | "b" | "c" | ... | "z"
<idstart> ::= <alphabet> | "_"
<idcont> ::= <alphabet> | "_" | <digit>
<keyword> ::= "False" | "None" | "True" | "and" | "break" | "continue" | "def"
            | "elif" | "else" | "except" | "for" | "from" | "if" | "in"
            | "lambda" | "not" | "or" | "pass" | "raise" | "return" | "try"
            | "while" | "yield"
<id> ::= <idstart> <idcont>* butnot <keyword>
<program> ::= <stmt>* <expr> // programs
<block> ::= <INDENT> <stmt>+ <DEDENT> | <stmt> // blocks
// statements
<stmt> ::= "pass" | <expr> | <id> "=" <expr> | <expr> "[" <expr> "]" "=" <expr>
           // if-statements
           | "if" <expr> ":" <block> [ "elif" <expr> ":" <block> ]*
           [ "else" ":" <block> ]
           // while-statements and for-statements
           | "while" <expr> ":" <block> | "for" <id> "in" <expr> ":" <block>
           // loop control statements, try statements, and raise statements
           | "break" | "continue" | "try" ":" <block> | "except" ":" <block>
           // function definitions
           | "def" <id> "(" [ <id> [ "," <id> ]* ] ")" ":" <block>
           // raise, return, yield, and yield from statements
           | "raise" | "return" <expr> | "yield" <expr> | "yield" "from" <expr>
// expressions
<expr> ::= "None" | <number> | "True" | "False" | <id>
           // unary/binary operators and parentheses
           | <uop> <expr> | <expr> <bop> <expr> | "(" <expr> ")"
           // lists
           | "[" [ <expr> [ "," <expr> ]* ] "]"
           // list append method and get item
           | <expr> "." "append" "(" <expr> ")" | <expr> "[" <expr> "]"
           // lambda functions
           | "lambda" [ <id> [ "," <id> ]* ] ":" <expr>
           // function applications
           | <expr> "(" [ <expr> [ "," <expr> ]* ] ")"
           // conditional expressions
           | <expr> "if" <expr> "else" <expr>
           // iter function and next function
           | "iter" "(" <expr> ")" | "next" "(" <expr> ")"
// operators
<uop> ::= "-" | "not"
<bop> ::= "+" | "-" | "*" | "//" | "%" | "and" | "or"
          | "==" | "!=" | "is" | "is not" | "<" | "<=" | ">" | ">="

```

The precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Method Call	<code>_.append</code>	10	left
Prefix Unary	<code>-</code>	9	right
Multiplicative	<code>*, //, %</code>	8	
Additive	<code>+, -</code>	7	
Comparison	<code>==, !=, is, is not, <, <=, >, >=</code>	6	
Logical Negation	<code>not</code>	5	
Logical Conjunction	<code>and</code>	4	
Logical Disjunction	<code>or</code>	3	
Conditional Expression	<code>if-else</code>	2	-
Lambda Function	<code>lambda</code>	1	-

3 ABSTRACT SYNTAX

The abstract syntax of MiniPython is defined as follows:

Programs	$p ::= S; \dots; S; e$	(Program)	Blocks	$B ::= S \{S; \dots; S\}$	(Block)
Statements	$S ::= \text{pass}$	(SPass)		$ \text{continue}$	(SContinue)
	$ e$	(SExpr)		$ \text{try } B \text{ except } B$	(STry)
	$ x = e$	(SAssign)		$ \text{raise}$	(SRaise)
	$ e[e] = e$	(SSetItem)		$ \text{def } x(x, \dots, x) B$	(SDef)
	$ \text{if } e B \text{ else } B$	(SIf)		$ \text{return } e$	(SReturn)
	$ \text{while } e B$	(SWhile)		$ \text{yield } e$	(SYield)
	$ \text{break}$	(SBreak)			
Expressions	$e ::= \text{None}$	(ENone)		$ e[e]$	(EGetItem)
	$ n$	(Enum)		$ \lambda(x, \dots, x).e$	(ELambda)
	$ b$	(EBool)		$ e(e, \dots, e)$	(EApp)
	$ x$	(EId)		$ e \text{ if } e \text{ else } e$	(ECond)
	$ e \oplus e$	(EBOp)		$ \text{iter}(e)$	(EIIter)
	$ [e, \dots, e]$	(EList)		$ \text{next}(e)$	(ENext)
	$ e.\text{append}(e)$	(EAppend)			

Operators $\oplus ::= + (\text{Add}) | * (\text{Mul}) | // (\text{Div}) | \% (\text{Mod}) | < (\text{Lt}) | <= (\text{Lte}) | == (\text{Eq}) | \text{is} (\text{Is})$

where $\begin{cases} \text{Booleans } \mathbb{B} \ni b ::= \text{True} | \text{False} & (\text{Boolean}) \\ \text{Identifiers } x \in \mathbb{X} & (\text{String}) \\ \text{Numbers } n, m, k \in \mathbb{Z} & (\text{BigInt}) \end{cases}$

The semantics of the remaining cases are defined with the following desugaring rules:

$$\begin{aligned}
 \mathcal{D}[-e] &= \mathcal{D}[e] * (-1) & \mathcal{D}[e_1 \text{ is not } e_2] &= \mathcal{D}[\text{not}(e_1 \text{ is } e_2)] \\
 \mathcal{D}[e_1 - e_2] &= \mathcal{D}[e_1] + \mathcal{D}[-e_2] & \mathcal{D}[e_1 \text{ and } e_2] &= \text{if } (\mathcal{D}[e_1]) \mathcal{D}[e_2] \text{ else False} \\
 \mathcal{D}[e_1 > e_2] &= \mathcal{D}[\text{not}(e_1 \leq e_2)] & \mathcal{D}[e_1 \text{ or } e_2] &= \text{if } (\mathcal{D}[e_1]) \text{ True else } \mathcal{D}[e_2] \\
 \mathcal{D}[e_1 \geq e_2] &= \mathcal{D}[\text{not}(e_1 < e_2)] & \mathcal{D}[\text{not } e] &= \text{if } (\mathcal{D}[e]) \text{ False else True} \\
 \mathcal{D}[e_1 != e_2] &= \mathcal{D}[\text{not}(e_1 == e_2)] & \mathcal{D}[\text{yield from } e] &= \mathcal{D}[\text{for } x \text{ in } e \text{ yield } x] \\
 \mathcal{D}[\text{if } e B \text{ elif } e_1 B_1 \dots \text{elif } e_n B_n \text{ else } B'] &= \mathcal{D}[\text{if } e B \text{ else if } e_1 B_1 \dots \text{else if } e_n B_n \text{ else } B'] \\
 \mathcal{D}[\text{if } e B \text{ elif } e_1 B_1 \dots \text{elif } e_n B_n] &= \mathcal{D}[\text{if } e B \text{ else if } e_1 B_1 \dots \text{else if } e_n B_n \text{ else pass}] \\
 \mathcal{D}[\text{for } x \text{ in } e B] &= \mathcal{D}[\underline{x} = e; \text{while True } \{\text{try } x = \text{next}(\underline{x}) \text{ except break; } B\}]
 \end{aligned}$$

where \underline{x} denotes a fresh temporary variable. All the omitted cases recursively apply the desugaring rule to their sub-expressions. For example, $\mathcal{D}[e_1 + e_2] = \mathcal{D}[e_1] + \mathcal{D}[e_2]$.

4 SEMANTICS

We use the following notations in the semantics:

States $\langle \kappa \parallel s \parallel H \parallel M \rangle \in \mathbb{K} \times \mathbb{S} \times \mathbb{H} \times \mathbb{M}$ (State) Continuations $\mathbb{K} \ni \kappa ::= \square \mid i :: \kappa$ (Cont)

Instructions	$i ::= (\sigma \vdash_B B) \quad (\text{IBlock})$	$ \text{jmp-if}[\psi] \quad (\text{IJmpIf})$
	$ (\sigma \vdash_S S) \quad (\text{IStmt})$	$ \text{jmp}[c] \quad (\text{IJmp})$
	$ (\sigma \vdash_e e) \quad (\text{IExpr})$	$ \text{raise}(\omega) \quad (\text{IRaise})$
	$ (\oplus) \quad (\text{IBop})$	$ \text{call}[n] \quad (\text{ICall})$
	$ \text{write}[a] \quad (\text{IWrite})$	$ \text{return} \quad (\text{IReturn})$
	$ \text{get-item} \quad (\text{IGetItem})$	$ \text{yield} \quad (\text{IYield})$
	$ \text{set-item} \quad (\text{ISetItem})$	$ \text{iter} \quad (\text{IIter})$
	$ \text{list}[n] \quad (\text{IList})$	$ \text{next} \quad (\text{INext})$
	$ \text{append} \quad (\text{IAppend})$	$ \text{drop} \quad (\text{IDrop})$

Value Stacks $s ::= \square \mid v :: s$ (Stack)

Control Handlers	$H \in \mathbb{H} = \mathbb{C} \xrightarrow{\text{fin}} \Psi$ (Handler)	Environments	$\sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{A}$ (Env)
Memories	$M \in \mathbb{M} = \mathbb{A} \xrightarrow{\text{fin}} \mathbb{V}$ (Mem)	Addresses	$a \in \mathbb{A}$ (Addr)

Errors $\omega ::= \text{RuntimeError} \mid \text{ZeroDivisionError} \mid \text{TypeError} \mid \text{IndexError}$
 $\mid \text{StopIteration} \mid \text{NameError}[x]$

Values	$v ::= \text{None} \quad (\text{NoneV})$	$ \langle \lambda(x, \dots, x).B, \sigma \rangle \quad (\text{CloV})$
	$ n \quad (\text{NumV})$	$ \langle \lambda*(x, \dots, x).B, \sigma \rangle \quad (\text{GenV})$
	$ b \quad (\text{BoolV})$	$ \langle \kappa \parallel s \parallel H \rangle \quad (\text{ContV})$
	$ a \quad (\text{AddrV})$	$ \text{iter}[a, n] \quad (\text{IterV})$
	$ [v, \dots, v] \quad (\text{ListV})$	

Continuation Values $\psi, \langle \kappa \parallel s \parallel H \rangle \in \Psi = \mathbb{K} \times \mathbb{S} \times \mathbb{H}$ (KValue)

Controls	$c ::= \text{return} \quad (\text{Return})$	$ \text{throw} \quad (\text{Throw})$
	$ \text{break} \quad (\text{Break})$	$ \text{finally} \quad (\text{Finally})$
	$ \text{continue} \quad (\text{Continue})$	$ \text{yield} \quad (\text{Yield})$

The small-step operational (reduction) semantics of MiniPython is defined in the following form of the reduction relation (\rightarrow):

$$\boxed{\langle \kappa \parallel s \parallel H \parallel M \rangle \rightarrow \langle \kappa \parallel s \parallel H \parallel M \rangle}$$

The evaluation result of a program $p = S_1; \dots; S_n; e$ is the value v if

$$\langle (\sigma \vdash S_1) :: \dots :: (\sigma \vdash S_n) :: (\sigma \vdash e) :: \square \parallel \square \parallel \emptyset \parallel \emptyset \rangle \rightarrow^* \langle \square \parallel v :: \square \parallel _ \parallel _ \rangle$$

$$\text{where } \left\{ \begin{array}{lcl} \{x_1, \dots, x_k\} & = & \text{locals}(\{S_1, \dots, S_n\}) \\ a_1, \dots, a_k & = & (\text{distinct fresh addresses}) \\ \sigma & = & [x_1 \mapsto a_1, \dots, x_k \mapsto a_k] \\ M & = & [a_1 \mapsto \text{None}, \dots, a_k \mapsto \text{None}] \end{array} \right.$$

4.1 Reduction Relations for IStmt

SPass	$\langle (\sigma \vdash_S \text{pass}) :: \kappa s H M \rangle$
\rightarrow	$\langle \kappa s H M \rangle$
SExpr	$\langle (\sigma \vdash_S e) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{drop} :: \kappa s H M \rangle$
SAssign	$\langle (\sigma \vdash_S x = e) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{write}[\sigma(x)] :: \kappa s H M \rangle$
SSetItem	$\langle (\sigma \vdash_S e_0[e_1] = e_2) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_2) :: (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \text{set-item} :: \kappa s H M \rangle$
SIf	$\langle (\sigma \vdash_S \text{if } e B_0 \text{ else } B_1) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{jmp-if}[\langle (\sigma \vdash_B B_0) :: \kappa s H \rangle] :: (\sigma \vdash_B B_1) :: \kappa s H M \rangle$
SWhile	$\langle (\sigma \vdash_S \text{while } e B) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{jmp-if}[\psi_{\text{body}}] :: \kappa s H M \rangle$
where	$\begin{cases} \psi_{\text{body}} = \langle (\sigma \vdash_B B) :: (\sigma \vdash_S \text{while } e B) :: \kappa s H_{\text{body}} \rangle \\ H_{\text{body}} = H[\text{continue} \mapsto \psi_{\text{continue}}, \text{break} \mapsto \psi_{\text{break}}] \\ \psi_{\text{continue}} = \langle (\sigma \vdash_S \text{while } e B) :: \kappa s H \rangle \\ \psi_{\text{break}} = \langle \kappa s H \rangle \end{cases}$
SBreak	$\langle (\sigma \vdash_S \text{break}) :: \kappa s H M \rangle$
\rightarrow	$\langle \text{jmp}[\text{break}] :: \kappa s H M \rangle$
SContinue	$\langle (\sigma \vdash_S \text{continue}) :: \kappa s H M \rangle$
\rightarrow	$\langle \text{jmp}[\text{continue}] :: \kappa s H M \rangle$
STry	$\langle (\sigma \vdash_S \text{try } B_0 \text{ except } B_1) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_B B_0) :: \text{jmp}[\text{finally}] :: \square s H_{\text{body}} M \rangle$
where	$\begin{cases} H_{\text{body}} = H[\text{raise} \mapsto \psi_{\text{raise}}, \text{finally} \mapsto \psi_{\text{finally}}] \\ \psi_{\text{raise}} = \langle (\sigma \vdash_B B_1) :: \kappa s H \rangle \\ \psi_{\text{finally}} = \langle \kappa s H \rangle \end{cases}$
SRaise	$\langle (\sigma \vdash_S \text{raise}) :: \kappa s H M \rangle$
\rightarrow	$\langle \text{raise}(\text{RuntimeError}) :: \square s H M \rangle$
SDef	$\langle (\sigma \vdash_S \text{def } x_0 (x_1, \dots, x_n). B) :: \kappa s H M \rangle$
\rightarrow	$\langle \text{write}[\sigma(x_0)] :: \kappa a :: s H M[a \mapsto v] \rangle$
where	$a \notin \text{Domain}(M) \text{ and } v = \begin{cases} \langle \lambda*(x_1, \dots, x_n). B, \sigma \rangle & \text{if hasYield}(B) \\ \langle \lambda(x_1, \dots, x_n). B, \sigma \rangle & \text{otherwise} \end{cases}$
SReturn	$\langle (\sigma \vdash_S \text{return } e) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{return} :: \kappa s H M \rangle$
SYield	$\langle (\sigma \vdash_S \text{yield } e) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{yield} :: \kappa s H M \rangle$

4.2 Reduction Relations for IBlock

$$\begin{array}{lcl} \text{IBlock} & \langle (\sigma \vdash_B \{S_1; \dots; S_n\}) :: \kappa || s || H || M \rangle \\ \rightarrow & \langle (\sigma \vdash_S S_1) :: \dots :: (\sigma \vdash_S S_n) :: \kappa || s || H || M \rangle \end{array}$$

4.3 Reduction Relations for IExpr

<code>ENone</code>	$\langle (\sigma \vdash_e \text{None}) :: \kappa s H M \rangle$
\rightarrow	$\langle \kappa \text{None} :: s H M \rangle$
<code>Enum</code>	$\langle (\sigma \vdash_e n) :: \kappa s H M \rangle$
\rightarrow	$\langle \kappa n :: s H M \rangle$
<code>EBool</code>	$\langle (\sigma \vdash_e b) :: \kappa s H M \rangle$
\rightarrow	$\langle \kappa b :: s H M \rangle$
<code>EId</code>	$\langle (\sigma \vdash_e x) :: \kappa s H M \rangle$
\rightarrow	$\begin{cases} \langle \kappa M(\sigma(x)) :: s H M \rangle & \text{if } x \in \text{Domain}(\sigma) \\ \langle \text{raise}(NameError[x]) :: \square s H M \rangle & \text{otherwise} \end{cases}$
<code>EBOp</code>	$\langle (\sigma \vdash_e e_0 \oplus e_1) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: (\oplus) :: \kappa s H M \rangle$
<code>EList</code>	$\langle (\sigma \vdash_e [e_1, \dots, e_n]) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_1) :: \dots :: (\sigma \vdash_e e_n) :: \text{list}[n] :: \kappa s H M \rangle$
<code>EAppend</code>	$\langle (\sigma \vdash_e e_0.\text{append}(e_1)) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \text{append} :: \kappa s H M \rangle$
<code>EGetItem</code>	$\langle (\sigma \vdash_e e_0[e_1]) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \text{get-item} :: \kappa s H M \rangle$
<code>ELambda</code>	$\langle (\sigma \vdash_e \lambda(x_1, \dots, x_n).e) :: \kappa s H M \rangle$
\rightarrow	$\langle \kappa a :: s H M[a \mapsto \langle \lambda(x_1, \dots, x_n).\text{return } e, \sigma \rangle] \rangle$ where $a \notin \text{Domain}(M)$
<code>EApp</code>	$\langle (\sigma \vdash_e e_0(e_1, \dots, e_n)) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_0) :: (\sigma \vdash_e e_1) :: \dots :: (\sigma \vdash_e e_n) :: \text{call}[n] :: \kappa s H M \rangle$
<code>ECond</code>	$\langle (\sigma \vdash_e e_0 \text{ if } e_1 \text{ else } e_2) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e_1) :: \text{jmp-if}[\psi] :: (\sigma \vdash_e e_2) :: \kappa s H M \rangle$ where $\psi = \langle (\sigma \vdash_e e_0) :: \kappa s H \rangle$
<code>EIter</code>	$\langle (\sigma \vdash_e \text{iter}(e)) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{iter} :: \kappa s H M \rangle$
<code>ENext</code>	$\langle (\sigma \vdash_e \text{next}(e)) :: \kappa s H M \rangle$
\rightarrow	$\langle (\sigma \vdash_e e) :: \text{next} :: \kappa s H M \rangle$

4.4 Reduction Relations for IBop

Add	$\langle (+) :: \kappa n_2 :: n_1 :: s H M \rangle$	$\rightarrow \langle \kappa (n_1 + n_2) :: s H M \rangle$
Mul	$\langle (*) :: \kappa n_2 :: n_1 :: s H M \rangle$	$\rightarrow \langle \kappa (n_1 \times n_2) :: s H M \rangle$
Div ₀	$\langle (/) :: \kappa 0 :: n_1 :: s H M \rangle$	$\rightarrow \langle \text{raise}(ZeroDivisionError) :: \square s H M \rangle$
Div	$\langle (/) :: \kappa n_2 :: n_1 :: s H M \rangle$	$\rightarrow \langle \kappa (n_1 \div n_2) :: s H M \rangle$
Mod ₀	$\langle (\%) :: \kappa 0 :: n_1 :: s H M \rangle$	$\rightarrow \langle \text{raise}(ZeroDivisionError) :: \square s H M \rangle$
Mod	$\langle (\%) :: \kappa n_2 :: n_1 :: s H M \rangle$	$\rightarrow \langle \kappa (n_1 \% n_2) :: s H M \rangle$
Eq	$\langle (==) :: \kappa v_2 :: v_1 :: s H M \rangle$	$\rightarrow \langle \kappa \text{equal}(v_1, v_2, M) :: s H M \rangle$
Is	$\langle (\text{is}) :: \kappa v_2 :: v_1 :: s H M \rangle$	$\rightarrow \langle \kappa \text{is}(v_1, v_2) :: s H M \rangle$

$$\begin{aligned} \text{Lt} & \quad \langle (<) :: \kappa || v_2 :: v_1 :: s || H || M \rangle \\ & \rightarrow \begin{cases} \langle \kappa || b :: s || H || M \rangle & \text{if } \text{lessThan}(v_1, v_2, M) = b \\ \langle \text{raise}(TypeError) :: \square || s || H || M \rangle & \text{otherwise} \end{cases} \\ \text{Lte} & \quad \langle (<=) :: \kappa || v_2 :: v_1 :: s || H || M \rangle \\ & \rightarrow \begin{cases} \langle \kappa || (b \vee \text{equal}(v_1, v_2, M)) :: s || H || M \rangle & \text{if } \text{lessThan}(v_1, v_2, M) = b \\ \langle \text{raise}(TypeError) :: \square || s || H || M \rangle & \text{otherwise} \end{cases} \end{aligned}$$

4.5 Reduction Relations for Other Instructions

IWrite	$\langle \text{write}[a] :: \kappa v :: s H M \rangle$	$\rightarrow \langle \kappa s H M + (a \mapsto v) \rangle$
IGetItem	$\langle \text{get-item} :: \kappa n :: a :: s H M \rangle$	$\rightarrow \begin{cases} \langle \kappa v_{m+n} :: s H M \rangle & \text{if } M(a) = [v_1, \dots, v_m] \wedge -m \leq n < 0 \\ \langle \kappa v_n :: s H M \rangle & \text{if } M(a) = [v_1, \dots, v_m] \wedge 0 \leq n < m \\ \langle \text{raise}(IndexError) :: \square s H M \rangle & \text{if } M(a) = [v_1, \dots, v_m] \wedge (n < -m \vee m \leq n) \\ \langle \text{raise}(TypeError) :: \square s H M \rangle & \text{otherwise} \end{cases}$
ISetItem	$\langle \text{set-item} :: \kappa n :: a :: v :: s H M \rangle$	$\rightarrow \begin{cases} \langle \kappa s H M + (a \mapsto v'_{m+n}) \rangle & \text{if } M(a) = [v_1, \dots, v_m] \wedge -m \leq n < 0 \\ \langle \kappa s H M + (a \mapsto v'_n) \rangle & \text{if } M(a) = [v_1, \dots, v_m] \wedge 0 \leq n < m \\ \langle \text{raise}(IndexError) :: \square s H M \rangle & \text{if } M(a) = [v_1, \dots, v_m] \wedge (n < -m \vee m \leq n) \\ \langle \text{raise}(TypeError) :: \square s H M \rangle & \text{otherwise} \end{cases}$
		where $v'_k = [v_1, \dots, v_{k-1}, v, v_{k+1}, \dots, v_m]$

IList	$\langle \text{list}[n] :: \kappa v_n :: \dots :: v_1 :: s H M \rangle$
\rightarrow	$\langle \kappa a :: s H M + (a \mapsto [v_1, \dots, v_n]) \rangle$ where $a \notin \text{Domain}(M)$
IAppend	$\langle \text{append} :: \kappa v :: a :: s H M \rangle$
\rightarrow	$\begin{cases} \langle \kappa a :: s H M + (a \mapsto [v_1, \dots, v_m, v]) \rangle & \text{if } M(a) = [v_1, \dots, v_m] \\ \langle \text{raise}(\text{TypeError}) :: \square s H M \rangle & \text{otherwise} \end{cases}$
IJmpIf	$\langle \text{jmp-if}[\langle \kappa' s' H' \rangle] :: \kappa v :: s H M \rangle$
\rightarrow	$\begin{cases} \langle \kappa' s' H' M \rangle & \text{if } \text{isTruthy}(v, M) = \text{True} \\ \langle \kappa s H M \rangle & \text{otherwise} \end{cases}$
IJmp	$\langle \text{jmp}[c] :: \kappa s H M \rangle$
\rightarrow	$\langle \kappa' s' H' M \rangle$ where $\langle \kappa' s' H' \rangle = H(c)$
IRaise	$\langle \text{raise}(\omega) :: \kappa s H M \rangle$
\rightarrow	$\begin{cases} \langle \text{jmp}[\text{raise}] :: \square s H M \rangle & \text{if } \text{raise} \in \text{Domain}(H) \\ (\text{runtime error with message with } \omega\text{'s name}) & \text{otherwise} \end{cases}$
ICall	$\langle \text{call}[n] :: \kappa v_n :: \dots :: v_1 :: a :: s H M \rangle$
\rightarrow	$\begin{cases} \langle \kappa' \text{None} :: \blacksquare H_{\text{body}} M_2 \rangle & \text{if } M(a) = \langle \lambda(x_1, \dots, x_n).B, \sigma' \rangle \\ \langle \kappa a_{\text{iter}} :: s H M_4 \rangle & \text{if } M(a) = \langle \lambda*(x_1, \dots, x_n).B, \sigma' \rangle \\ \langle \text{raise}(\text{TypeError}) :: \square s H M \rangle & \text{otherwise} \end{cases}$
where	$\begin{cases} a_1, \dots, a_n \notin \text{Domain}(M) \\ \{x'_1, \dots, x'_m\} = \text{locals}(B) \setminus \{x_1, \dots, x_n\} \\ M_1 = M[a_1 \mapsto v_1, \dots, a_n \mapsto v_n] \\ a'_1, \dots, a'_m \notin \text{Domain}(M_1) \\ M_2 = M_1[a'_1 \mapsto \text{None}, \dots, a'_m \mapsto \text{None}] \\ \sigma_{\text{body}} = \sigma'[x_1 \mapsto a_1, \dots, x_n \mapsto a_n, x'_1 \mapsto a'_1, \dots, x'_m \mapsto a'_m] \\ H_{\text{body}} = H[\text{return} \mapsto \psi_{\text{return}}] \setminus \{\text{break}, \text{continue}, \text{yield}\} \\ \psi_{\text{return}} = \langle \kappa s H \rangle \\ \kappa' = (\sigma_{\text{body}} \vdash_B B) :: \text{return} :: \square \\ \psi_{\text{next}} = \langle \kappa' \text{None} :: \blacksquare H_{\text{body}} \rangle \\ a_{\text{cont}} \notin \text{Domain}(M_2) \\ M_3 = M_2[a_{\text{cont}} \mapsto \psi_{\text{next}}] \\ a_{\text{iter}} \notin \text{Domain}(M_3) \\ M_4 = M_3[a_{\text{iter}} \mapsto \text{iter}[a_{\text{cont}}, 0]] \end{cases}$
IReturn	$\langle \text{return} :: \kappa v :: s H M \rangle$
\rightarrow	$\langle \kappa' v :: s' H' M \rangle$ where $\langle \kappa' s' H' \rangle = H(\text{return})$
IYield	$\langle \text{yield} :: \kappa v :: s H M \rangle$
\rightarrow	$\langle \kappa' \langle \kappa s H \rangle :: v :: s' H' M \rangle$ where $\langle \kappa' s' H' \rangle = H(\text{yield})$

$$\begin{array}{ll}
 \text{IIter} & \langle \text{iter} :: \kappa \parallel a :: s \parallel H \parallel M \rangle \\
 \rightarrow & \begin{cases} \langle \kappa \parallel a :: s \parallel H \parallel M \rangle & \text{if } M(a) = \text{iter}[_, i] \\ \langle \kappa \parallel a_{\text{iter}} :: s \parallel H \parallel M[a_{\text{iter}} \mapsto \text{iter}[a, 0]] \rangle & \text{if } M(a) = [_] \\ \langle \text{raise}(TypeError) :: \square \parallel s \parallel H \parallel M \rangle & \text{otherwise} \end{cases} \\
 & \text{where } a_{\text{iter}} \notin \text{Domain}(M)
 \end{array}$$

$$\begin{array}{ll}
 \text{INext} & \langle \text{next} :: \kappa \parallel a :: s \parallel H \parallel M \rangle \\
 \rightarrow & \begin{cases} \langle \kappa' \parallel s' \parallel H_{\text{next}} \parallel M \rangle & \text{if } v = \text{iter}[a', m] \wedge v' = \langle \kappa' \parallel s' \parallel H' \rangle \\ \langle \kappa \parallel v_m :: s \parallel H \parallel M_{\text{next}} \rangle & \text{if } v = \text{iter}[a', m] \wedge v' = [v_1, \dots, v_n] \wedge m < n \\ \langle \text{raise}(StopIteration) :: \square \parallel s \parallel H \parallel M \rangle & \text{if } v = \text{iter}[a', m] \wedge v' = [v_1, \dots, v_n] \wedge n \leq m \\ \langle \text{raise}(TypeError) :: \square \parallel s \parallel H \parallel M \rangle & \text{otherwise} \end{cases} \\
 & \text{where } \begin{cases} v & = M(a) \\ v' & = M(a') \\ \psi_{\text{yield}} & = \langle \text{write}[a'] :: \kappa \parallel s \parallel H \rangle \\ \psi_{\text{return}} & = \langle \text{drop} :: \text{raise}(StopIteration) :: \square \parallel s \parallel H \rangle \\ H_{\text{next}} & = H'[y \mapsto \psi_{\text{yield}}, r \mapsto \psi_{\text{return}}] \\ M_{\text{next}} & = M[a \mapsto \text{iter}[a', m + 1]] \end{cases}
 \end{array}$$

$$\begin{array}{ll}
 \text>IDrop} & \langle \text{drop} :: \kappa \parallel v :: s \parallel H \parallel M \rangle \\
 \rightarrow & \langle \kappa \parallel s \parallel H \parallel M \rangle
 \end{array}$$

Other remaining cases raise a `TypeError` as follow:

$$\langle \kappa \parallel s \parallel H \parallel M \rangle \rightarrow \langle \text{raise}(TypeError) :: \square \parallel s \parallel H \parallel M \rangle$$

4.6 Auxiliary Definitions

The following auxiliary functions are used in the reduction rules. Note that $A \rightarrow B$ denotes a partial function from set A to set B .

$$\begin{array}{l}
 \boxed{\text{equal} : (\mathbb{V} \times \mathbb{V} \times \mathbb{M}) \rightarrow \mathbb{B}} \\
 \text{equal}(v, v', M) = \begin{cases} \text{True} & \text{if } \text{is}(v, v') \\ \text{equal}(M(a), M(a'), M) & \text{if } v = a \wedge v' = a' \\ \text{equal}(v_1, v'_1, M) \wedge \dots \wedge \text{equal}(v_n, v'_n, M) & \text{if } v = [v_1, \dots, v_n] \wedge v' = [v'_1, \dots, v'_n] \\ \text{False} & \text{otherwise} \end{cases} \\
 \boxed{\text{lessThan} : (\mathbb{V} \times \mathbb{V} \times \mathbb{M}) \rightarrow \mathbb{B}} \\
 \begin{array}{ll}
 \text{lessThan}(n_0, n_1, M) & = n_0 < n_1 \\
 \text{lessThan}(a_0, a_1, M) & = \text{lessThan}(M(a_0), M(a_1), M)
 \end{array} \\
 \text{lessThan}([v_1, \dots, v_n], [v'_1, \dots, v'_m], M) = \begin{cases} \text{False} & \text{if } n = m = 0 \\ \text{True} & \text{if } n = 0 < m \\ \text{False} & \text{if } n > 0 = m \\ \text{True} & \text{if } b \\ \text{False} & \text{if } \neg b \wedge \neg b' \\ \text{lessThan}([v_2, \dots, v_n], [v'_2, \dots, v'_m], M) & \text{if } \neg b \wedge b' \end{cases} \\
 \text{where } b = \text{lessThan}(v_1, v'_1, M) \text{ and } b' = \text{equal}(v_1, v'_1, M)
 \end{array}$$

$$\text{is} : (\mathbb{V} \times \mathbb{V}) \rightarrow \mathbb{B}$$

$$\text{is}(v, v') = \begin{cases} \text{True} & \text{if } v = v' = \text{None} \\ \text{True} & \text{if } v = v' = n \\ \text{True} & \text{if } v = v' = b \\ \text{True} & \text{if } v = v' = a \\ \text{False} & \text{otherwise} \end{cases}$$

$$\text{isTruthy} : (\mathbb{V} \times \mathbb{M}) \rightarrow \mathbb{B}$$

$$\text{isTruthy}(v, M) = \begin{cases} \text{False} & \text{if } v = \text{None} \\ n \neq 0 & \text{if } v = n \\ b & \text{if } v = b \\ \text{isTruthy}(M(a), M) & \text{if } v = a \\ 0 < n & \text{if } v = [v_1, \dots, v_n] \\ \text{True} & \text{otherwise} \end{cases}$$

$$\text{locals}(B) : \mathcal{P}(\mathbb{X})$$

$$\text{locals}(\{S_1; \dots; S_n\}) = \text{locals}(S_1) \cup \dots \cup \text{locals}(S_n)$$

$$\text{locals}(S) = \begin{cases} \{x\} & \text{if } S = x = e \\ \{x_0\} & \text{if } S = \text{def } x_0 (x_1, \dots, x_n) B \\ \text{locals}(B_0) \cup \text{locals}(B_1) & \text{if } S = \text{if } e B_0 \text{ else } B_1 \\ \text{locals}(B_0) \cup \text{locals}(B_1) & \text{if } S = \text{try } B_0 \text{ except } B_1 \\ \text{locals}(B) & \text{if } S = \text{while } e B \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{hasYield}(B) : \mathbb{B}$$

$$\text{hasYield}(\{S_1; \dots; S_n\}) = \text{hasYield}(S_1) \vee \dots \vee \text{hasYield}(S_n)$$

$$\text{hasYield}(S) = \begin{cases} \text{True} & \text{if } S = \text{yield } e \\ \text{hasYield}(B) & \text{if } S = \text{def } x_0 (x_1, \dots, x_n) B \\ \text{hasYield}(B_0) \vee \text{hasYield}(B_1) & \text{if } S = \text{if } e B_0 \text{ else } B_1 \\ \text{hasYield}(B_0) \vee \text{hasYield}(B_1) & \text{if } S = \text{try } B_0 \text{ except } B_1 \\ \text{hasYield}(B) & \text{if } S = \text{while } e B \\ \text{False} & \text{otherwise} \end{cases}$$