SICPy

Xin Zhe Khooi

2021

SICPy §0

module	::=	*statement* ...	statement sequence
statement	::=	*expression*	expression statement
expression	::=	*number*	primitive number expression
	|	`True` | `False`	primitive boolean expression
	|	*string*	primitive string expression
	|	*expression binary-operator expression*	binary operator combination
	|	*unary-operator expression*	unary operator combination
	|	(*expression*)	parenthesised expression
binary-operator	::=	`+` | `-` | `*` | `/` | `%` | `==`	
	|	`>` | `<` | `>=` | `<=` | `and` | `or`	
unary-operator	::=	`not` | `+` | `-`	

SICPy §1

$$
\begin{array}{rcll}
\textit{module} & ::= & \textit{statement} \ldots & \text{statement sequence} \\
\textit{statement} & ::= & \textit{name} = \textit{expression} & \text{assignment statement} \\
& | & \textit{function} & \text{function declaration} \\
& | & \underline{\texttt{return}}\ \textit{expression} & \text{return statement} \\
& | & \textit{if-statement} & \text{conditional statement} \\
& | & \textit{expression} & \text{expression statement} \\
\textit{function} & ::= & \underline{\texttt{def}}\ \textit{name}\ (\ \textit{parameters}\ )\ : & \\
& & \quad \textit{statement} \ldots & \text{function declaration} \\
\textit{parameters} & ::= & \epsilon \mid \textit{name}\ [\ ,\ \textit{name}\ ]\ \ldots & \text{function parameters} \\
\textit{if-statement} & ::= & \underline{\texttt{if}}\ \textit{expression}\ : & \\
& & \quad \textit{statement} \ldots & \\
& & [\ \underline{\texttt{elif}}\ \textit{expression}\ : & \\
& & \quad \textit{statement} \ldots & \\
& & [\ \underline{\texttt{elif}}\ \textit{expression}\ : & \\
& & \quad \textit{statement} \ldots\ \ ]\ \ldots & \\
& & \underline{\texttt{else}}\ : & \\
& & \quad \textit{statement} \ldots\ \ ] & \text{conditional statement} \\
\textit{expression} & ::= & \textit{number} & \text{primitive number expression} \\
& | & \texttt{True} \mid \texttt{False} & \text{primitive boolean expression} \\
& | & \textit{string} & \text{primitive string expression} \\
& | & \textit{name} & \text{name expression} \\
& | & \textit{expression binary-operator expression} & \text{binary operator combination} \\
& | & \textit{unary-operator expression} & \text{unary operator combination} \\
& | & \textit{expression}\ (\ \textit{expressions}\ ) & \text{function application} \\
& | & \underline{\texttt{lambda}}\ \textit{name}\ [\ ,\ \textit{name}\ ]\ \ldots: \textit{expression} & \text{lambda expression} \\
& | & \textit{expression}\ \underline{\texttt{if}}\ \textit{expression}\ \underline{\texttt{else}}\ \textit{expression} & \text{conditional expression} \\
& | & (\ \textit{expression}\ ) & \text{parenthesised expression} \\
\textit{binary-operator} & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{==} & \\
& | & \texttt{>} \mid \texttt{<} \mid \texttt{>=} \mid \texttt{<=} \mid \texttt{and} \mid \texttt{or} & \\
\textit{unary-operator} & ::= & \texttt{not} \mid \texttt{+} \mid \texttt{-} & \\
\textit{expressions} & ::= & \epsilon \mid \textit{expression}\ [\ ,\ \textit{expression}\ ]\ \ldots & \text{argument expressions} \\
\end{array}
$$

SICPy §2

module	::=	*statement* ...	statement sequence
statement	::=	*name* = *expression*	assignment statement
	|	*function*	function declaration
	|	<u>return</u> *expression*	return statement
	|	*if-statement*	conditional statement
	|	*expression*	expression statement
function	::=	<u>def</u> *name* (*parameters*) :	
		statement ...	function declaration
parameters	::=	ϵ | *name* [, *name*] ...	function parameters
if-statement	::=	<u>if</u> *expression* :	
		statement ...	
	[	<u>elif</u> *expression* :	
		statement ...	
	[	<u>elif</u> *expression* :	
		statement ...] ...	
		<u>else</u> :	
		statement ...]	conditional statement
expression	::=	*number*	primitive number expression
	|	<u>True</u> | <u>False</u>	primitive boolean expression
	|	<u>None</u>	primitive list expression
	|	*string*	primitive string expression
	|	*name*	name expression
	|	*expression binary-operator expression*	binary operator combination
	|	*unary-operator expression*	unary operator combination
	|	*expression* (*expressions*)	function application
	|	<u>lambda</u> *name* [, *name*] ... : *expression*	lambda expression
	|	*expression* <u>if</u> *expression* <u>else</u> *expression*	conditional expression
	|	(*expression*)	parenthesised expression
binary-operator	::=	+ | - | * | / | % | ==	
	|	> | < | >= | <= | and | or	
unary-operator	::=	not | + | -	
expressions	::=	ϵ | *expression* [, *expression*] ...	argument expressions

SICPy §3

module	::=	*statement* ...	statement sequence
statement	::=	*name* = *expression*	assignment statement
	|	*function*	function declaration
	|	<u>return</u> *expression*	return statement
	|	*if-statement*	conditional statement
	|	*while-statement*	while statement
	|	*for-statement*	for statement
	|	*expression*	expression statement
	|	<u>break</u> | <u>pass</u> | <u>continue</u>	

function	::=	<u>def</u> *name* (*parameters*) :
		statement ... function declaration
parameters	::=	ϵ | *name* [, *name*] ... function parameters

if-statement	::=	<u>if</u> *expression* :
		statement ...
	[	<u>elif</u> *expression* :
		statement ...
	[	<u>elif</u> *expression* :
		statement ...] ...
		<u>else</u> :
		statement ...] conditional statement

while-statement	::=	<u>while</u> *expression* :
		statement ... while statement

for-statement	::=	<u>for</u> *expression* <u>in</u> *expression* :
		statement ... for statement

expression	::=	*number*	primitive number expression
	|	<u>True</u> | <u>False</u>	primitive boolean expression
	|	<u>None</u>	primitive list expression
	|	*string*	primitive string expression
	|	*name*	name expression
	|	*expression* *binary-operator* *expression*	binary operator combination
	|	*unary-operator* *expression*	unary operator combination
	|	*expression* (*expressions*)	function application
	|	<u>lambda</u> *name* [, *name*] ... : *expression*	lambda expression
	|	*expression* <u>if</u> *expression* <u>else</u> *expression*	conditional expression
	|	*list-expression*	list expression
	|	*expression* [*expression*]	list access
	|	(*expression*)	parenthesised expression

list-expression	::=	[*expressions*]	literal list expression
	|	[*expression* <u>for</u> *expression* <u>in</u> *expression* [if *expression*]]	list comprehension expression
binary-operator	::=	+ | - | * | / | % | ==	
	|	> | < | >= | <= | and | or	
unary-operator	::=	not | + | -	
expressions	::=	ϵ | *expression* [, *expression*] ...	argument expressions

SICPy §4

module	::=	*statement* ...	statement sequence
statement	::=	*name* = *expression*	single assignment
	|	*name* [, *name*] ... = *expression*	tuple assignment
	|	*function*	function declaration
	|	<u>return</u> *expression*	return statement
	|	*if-statement*	conditional statement
	|	*while-statement*	while statement
	|	*for-statement*	for statement
	|	*try-statement*	try statement
	|	*expression*	expression statement
	|	<u>break</u> | <u>pass</u> | <u>continue</u>	
function	::=	<u>def</u> *name* (*parameters*) :	
		statement ...	function declaration
parameters	::=	ϵ | *name* [, *name*] ...	function parameters
if-statement	::=	<u>if</u> *expression* :	
		statement ...	
	[	<u>elif</u> *expression* :	
		statement ...	
	[	<u>elif</u> *expression* :	
		statement ...] ...	
		<u>else</u> :	
		statement ...]	conditional statement
while-statement	::=	<u>while</u> *expression* :	
		statement ...	while statement
for-statement	::=	<u>for</u> *expression* <u>in</u> *expression* :	
		statement ...	for statement
try-statement	::=	<u>try</u> :	
		statement ...	
		<u>except</u> *expression* :	
		statement ...	try statement
expression	::=	*number*	primitive number expression
	|	<u>True</u> | <u>False</u>	primitive boolean expression
	|	<u>None</u>	primitive list expression
	|	*string*	primitive string expression
	|	*name*	name expression
	|	*expression binary-operator expression*	binary operator combination
	|	*unary-operator expression*	unary operator combination
	|	*expression* (*expressions*)	function application
	|	<u>lambda</u> *name* [, *name*] ...: *expression*	lambda expression
	|	*expression* <u>if</u> *expression* <u>else</u> *expression*	conditional expression
	|	*list-expression*	list expression
	|	{ *expression* : *expression* [, *expression* : *expression*] ... }	literal dict expression
	|	(*tuple-expression*)	tuple expression
	|	*expression* [*expression*]	list/ dictionary access
	|	(*expression*)	parenthesised expression

$$
\begin{array}{rcll}
\textit{list-expression} & ::= & \texttt{[} \textit{expressions} \texttt{]} & \text{literal list expression} \\
& | & \texttt{[} \textit{expression} \underline{\texttt{for}} \textit{ expression } \underline{\texttt{in}} \textit{ expression } [\texttt{ if } \textit{expression}] \texttt{]} & \text{list comprehension expression} \\
\textit{tuple-expression} & ::= & \epsilon \mid \textit{expression} \texttt{ , } \textit{expressions} & \textit{tuple expression} \\
\textit{binary-operator} & ::= & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\%} \mid \texttt{==} \\
& | & \texttt{>} \mid \texttt{<} \mid \texttt{>=} \mid \texttt{<=} \mid \texttt{and} \mid \texttt{or} \\
\textit{unary-operator} & ::= & \texttt{not} \mid \texttt{+} \mid \texttt{-} \\
\textit{expressions} & ::= & \epsilon \mid \textit{expression} [\texttt{ , } \textit{expression}] \ldots & \text{argument expressions}
\end{array}
$$