

# SICPy Specifications

Khooi Xin Zhe, Ang Jun Jie

19 April, 2021

## SICPy §0

<i>module</i>	::=	<i>statement ...</i>	statement sequence
<i>statement</i>	::=	<i>expression</i>	expression statement
<i>expression</i>	::=	<i>number</i>	primitive number expression
		<u><b>True</b></u>   <u><b>False</b></u>	primitive boolean expression
		<i>string</i>	primitive string expression
		<i>expression binary-operator expression</i>	binary operator combination
		<i>unary-operator expression</i>	unary operator combination
		( <i>expression</i> )	parenthesised expression
<i>binary-operator</i>	::=	<b>+</b>   <b>-</b>   <b>*</b>   <b>/</b>   <b>%</b>   <b>==</b>	
		<b>&gt;</b>   <b>&lt;</b>   <b>&gt;=</b>   <b>&lt;=</b>   <b>and</b>   <b>or</b>	
<i>unary-operator</i>	::=	<b>not</b>   <b>+</b>   <b>-</b>	

## SICPy §1

<i>module</i>	<code>::= statement ...</code>	statement sequence
<i>statement</i>	<code>::= name = expression</code> <code>  function</code> <code>  <u>return</u> expression</code> <code>  if-statement</code> <code>  expression</code>	assignment statement function declaration return statement conditional statement expression statement
<i>function</i>	<code>::= <u>def</u> name ( parameters ) :</code> <code>statement ...</code>	function declaration
<i>parameters</i>	<code>::= <math>\epsilon</math>   name [ , name ] ...</code>	function parameters
<i>if-statement</i>	<code>::= <u>if</u> expression :</code> <code>statement ...</code> <code>[[ <u>elif</u> expression :</code> <code>statement ... ] ...</code> <code><u>else</u> :</code> <code>statement ... ]</code>	conditional statement
<i>expression</i>	<code>::= number</code> <code>  <u>True</u>   <u>False</u></code> <code>  string</code> <code>  name</code> <code>  expression binary-operator expression</code> <code>  unary-operator expression</code> <code>  expression ( expressions )</code> <code>  <u>lambda</u> name [ , name ] ... : expression</code> <code>  expression <u>if</u> expression <u>else</u> expression</code> <code>  ( expression )</code>	primitive number expression primitive boolean expression primitive string expression name expression binary operator combination unary operator combination function application lambda expression conditional expression parenthesised expression
<i>binary-operator</i>	<code>::= +   -   *   /   %   ==</code> <code>  &gt;   &lt;   &gt;=   &lt;=   and   or</code>	
<i>unary-operator</i>	<code>::= not   +   -</code>	
<i>expressions</i>	<code>::= <math>\epsilon</math>   expression [ , expression ] ...</code>	argument expressions

## SICPy §2

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

## SICPy §3

<i>module</i> ::= <i>statement</i> ...	statement sequence
<i>statement</i> ::= <i>name</i> = <i>expression</i>	assignment statement
<i>function</i>	function declaration
<u>return</u> <i>expression</i>	return statement
<i>if-statement</i>	conditional statement
<i>while-statement</i>	while statement
<i>for-statement</i>	for statement
<i>expression</i>	expression statement
<u>break</u>   <u>pass</u>   <u>continue</u>	
<i>function</i> ::= <u>def</u> <i>name</i> ( <i>parameters</i> ) :	function declaration
<i>statement</i> ...	function parameters
<i>parameters</i> ::= $\epsilon$   <i>name</i> [ , <i>name</i> ] ...	
<i>if-statement</i> ::= <u>if</u> <i>expression</i> :	
<i>statement</i> ...	
[[ <u>elif</u> <i>expression</i> :	
<i>statement</i> ... ] ...	
<u>else</u> :	
<i>statement</i> ... ]	conditional statement
<i>while-statement</i> ::= <u>while</u> <i>expression</i> :	
<i>statement</i> ...	while statement
<i>for-statement</i> ::= <u>for</u> <i>expression</i> <u>in</u> <i>expression</i> :	
<i>statement</i> ...	for statement
<i>expression</i> ::= <i>number</i>	primitive number expression
<u>True</u>   <u>False</u>	primitive boolean expression
<u>None</u>	primitive list expression
<i>string</i>	primitive string expression
<i>name</i>	name expression
<i>expression</i> <i>binary-operator</i> <i>expression</i>	binary operator combination
<i>unary-operator</i> <i>expression</i>	unary operator combination
<i>expression</i> ( <i>expressions</i> )	function application
<u>lambda</u> <i>name</i> [ , <i>name</i> ] ... : <i>expression</i>	lambda expression
<i>expression</i> <u>if</u> <i>expression</i> <u>else</u> <i>expression</i>	conditional expression
<i>list-expression</i>	list expression
<i>expression</i> [ <i>expression</i> ]	list access
( <i>expression</i> )	parenthesised expression
<i>list-expression</i> ::= [ <i>expressions</i> ]	literal list expression
[ <i>expression</i> <u>for</u> <i>expression</i> <u>in</u> <i>expression</i> [ <u>if</u> <i>expression</i> ] ]	list comprehension expression
<i>binary-operator</i> ::= +   -   *   /   %   ==	
>   <   >=   <=   and   or	
<i>unary-operator</i> ::= not   +   -	
<i>expressions</i> ::= $\epsilon$   <i>expression</i> [ , <i>expression</i> ] ...	argument expressions

## SICPy §4

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

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