Scheme Cheat Sheet

Created by Ben Cuan - CS61A Fall 2021

Resources

- **This cheat sheet: https://go.cs61a.org/ben-scheme**
- **Scheme Specification: https://cs61a.org/articles/scheme-spec/**
- **Built-In Procedures: https://cs61a.org/articles/scheme-builtins/**

Variables	**Scheme**	**Python**
Numbers	123	123
Booleans	#t, #f	True, False
Assignment	(define hippo 1) *<returns hippo>*	hippo = 1 *<returns None>*

Booleans	**Scheme**	**Python**
Operations	(+ 1 2) (- 3 1)	(1 + 2) (3 - 1)
And	(and (+ 1 2) 'hi)	(1 + 2) and 'hi'
Or	(or (* 3 4) '(1))	(3 * 4) or Link(1)
Not	(not (- 5 6))	not (5 - 6)
Truthy Values	0, (print 'hi), #t, (list 1), nil, '(), etc.	'hi', -1, [3, 5], etc.
Falsey Values	#f	0, False, [], None, etc.
Comparing nums	(<= 7 8)	7 <= 8
Null check	(null? duck)	duck is None
Type checks	(<TYPE>? x) *<TYPE>: list, boolean, integer, atom...*	isinstance(x, <TYPE>) *<TYPE>: str, int, list, dict...*
Even/odd	(even? 61) (odd? 61)	61 % 2 == 0 61 % 2 == 1
Equals	(= a b) *<NUMBERS ONLY>*	a == b

	(eq? a b) *<NUMS/BOOLS/SYMBOLS>* (equal? a b) *<LISTS/PAIRS - checks if each element is equal>*	a is b *(not exact equivalence; see https://cs61a.org/articles/scheme-builtins/#general for more info)*

Functions	**Scheme**	**Python**

Function Definitions	(define (f x) (+ x 1))	def f(x):
		   return x + 1

Lambdas	(lambda (elephant) 7)	lambda elephant: 7

Higher order functions	(define (f x)
	  (define (g y) (+ x y))
	  g
	)
		def f(x):
		   def g(y):
		      return x + y
		   return g

Function calls	(define (f x) (+ x 1))
	
	(f 3)
		def f(x):
		   return x + 1
		f(3)

Control Statements	**Scheme**	**Python**

If	(if (< 4 5) 'yes 'no)	'yes' if (4 < 5) else 'no'
		
		- OR -
		if 4 < 5:
		    return 'yes'
		else:
		    return 'no'

Elif/Cond	(if (< a b) 1
	    (if (> a b) 2 3**)**)
	
	- OR -
	(cond
	   ((< a b) 1**)**
	   ((> a b) 2**)**
	   (else 3**)**
	)
		if a < b:
		  return 1
		elif a > b:
		  return 2
		else:
		  return 3

Begin
(Multi-line expressions)
	(begin
	    (print 'cs61a)
	    (print 'is_awesome!)
	)
		print('cs61a')
		print('is_awesome!')
		
		<python doesn't need begin, just type multiple lines!>

Let
(Temporary assignment)
	(let
	  ((x 1) (y 2)**)**
	  (+ x y)
	)
		(lambda x, y: x + y)(1, 2)
		
		<not a 1-1 correlation! let doesn't exist in python>

List Operations ***ALL SCHEME LISTS ARE LINKED LISTS!***	**Scheme**	**Python**
Create list	(cons first rest)	Link(first, rest)
Get value	(car lst)	lst.first
Get rest	(cdr lst)	lst.rest
Empty list	nil, '(), ()	Link.empty
Make long list	(list 1 2 3) *OR* '(1 2 3) *OR* (quote (1 2 3)) *OR* (cons 1 (cons 2 (cons 3 nil)))	Link(1, Link(2, Link(3, Link.empty)))
Map (Apply a function to every item in a list. Returns a new list.)	(map (lambda (b) (b*b)) '(1 2 3)) <returns '(1 4 9)> NOTE: "b" in the lambda function represents each item in the list, in order.	doubled = map(lambda b: b*b, [1, 2, 3, 4]) <doubled will be [1, 4, 9]> *OR* Pair(1, Pair(2, Pair(3, nil))).map(lambda b: b*b) <returns Pair(1, Pair(4, Pair(9, nil)))> NOTE: "b" in the lambda function represents each item in the list, in order.

Debugging Tips

Running Scheme in vscode

1. Install recommended extensions (how-to guide):
    a. Bracket Pair Colorizer
    b. vscode-scheme
2. Open **folder** containing Scheme assignment inside vscode (file->open folder)
3. Open terminal in vscode using [ctrl ~] (tilde is that key near esc)
4. Run `python3 scheme` in terminal
5. Enjoy!

Common Errors

- Unexpected EOF: most likely mismatched parentheses. Also, make sure you're calling functions like (f x) and not f(x).
- int is not callable: make sure you don't have parentheses around a number such as (0). Parentheses in Scheme treat the first element in the list as if it were a function; doing (0) in Python would look something like 0().
- incorrect number of arguments to cons: unlike Link in python, cons always needs exactly 2 arguments, a first and rest. If no rest is needed, put `nil`.