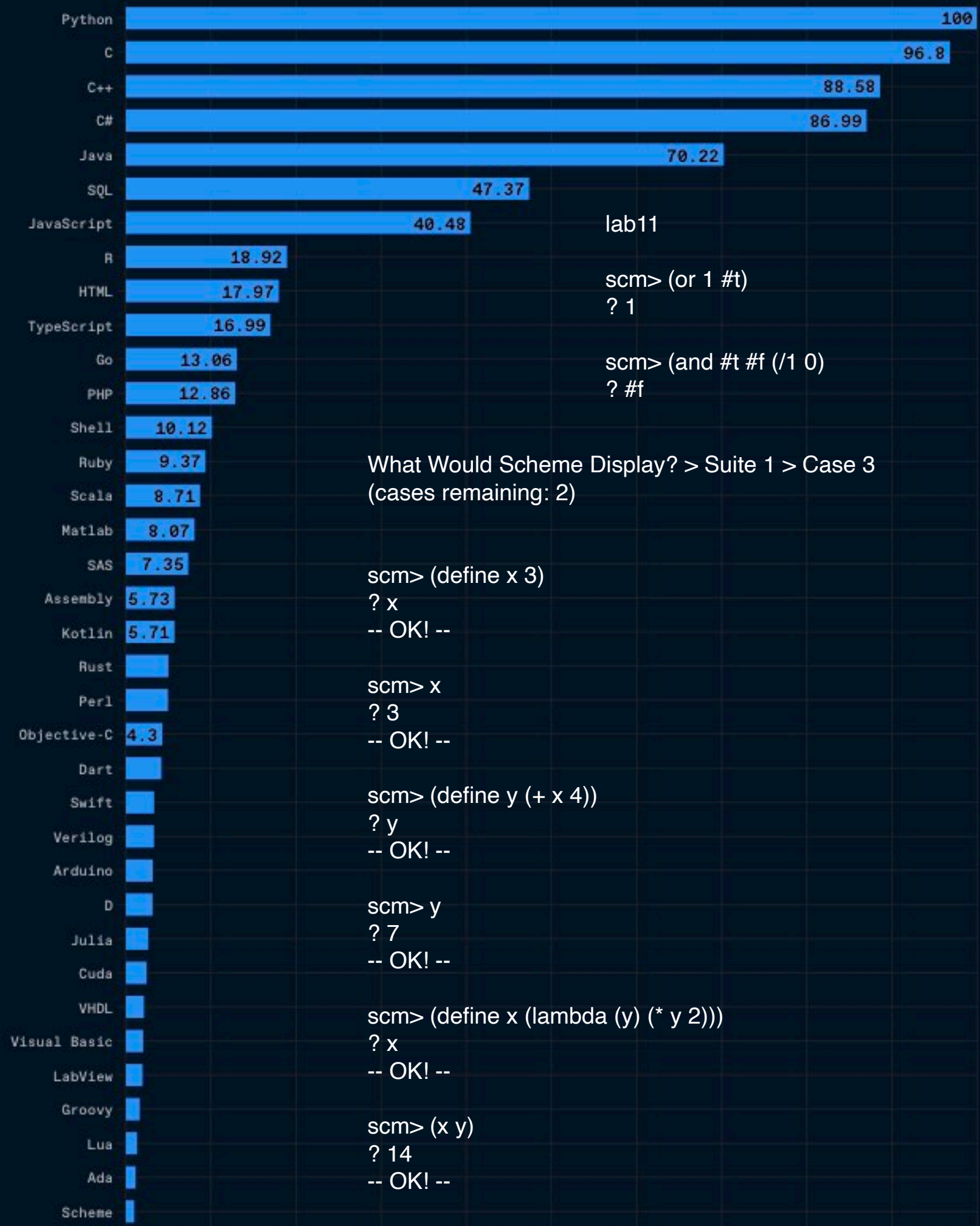


Programming Languages



lab11

scm> (or 1 #t)
? 1

scm> (and #t #f (/1 0))
? #f

What Would Scheme Display? > Suite 1 > Case 3
(cases remaining: 2)

scm> (define x 3)
? x
-- OK! --

scm> x
? 3
-- OK! --

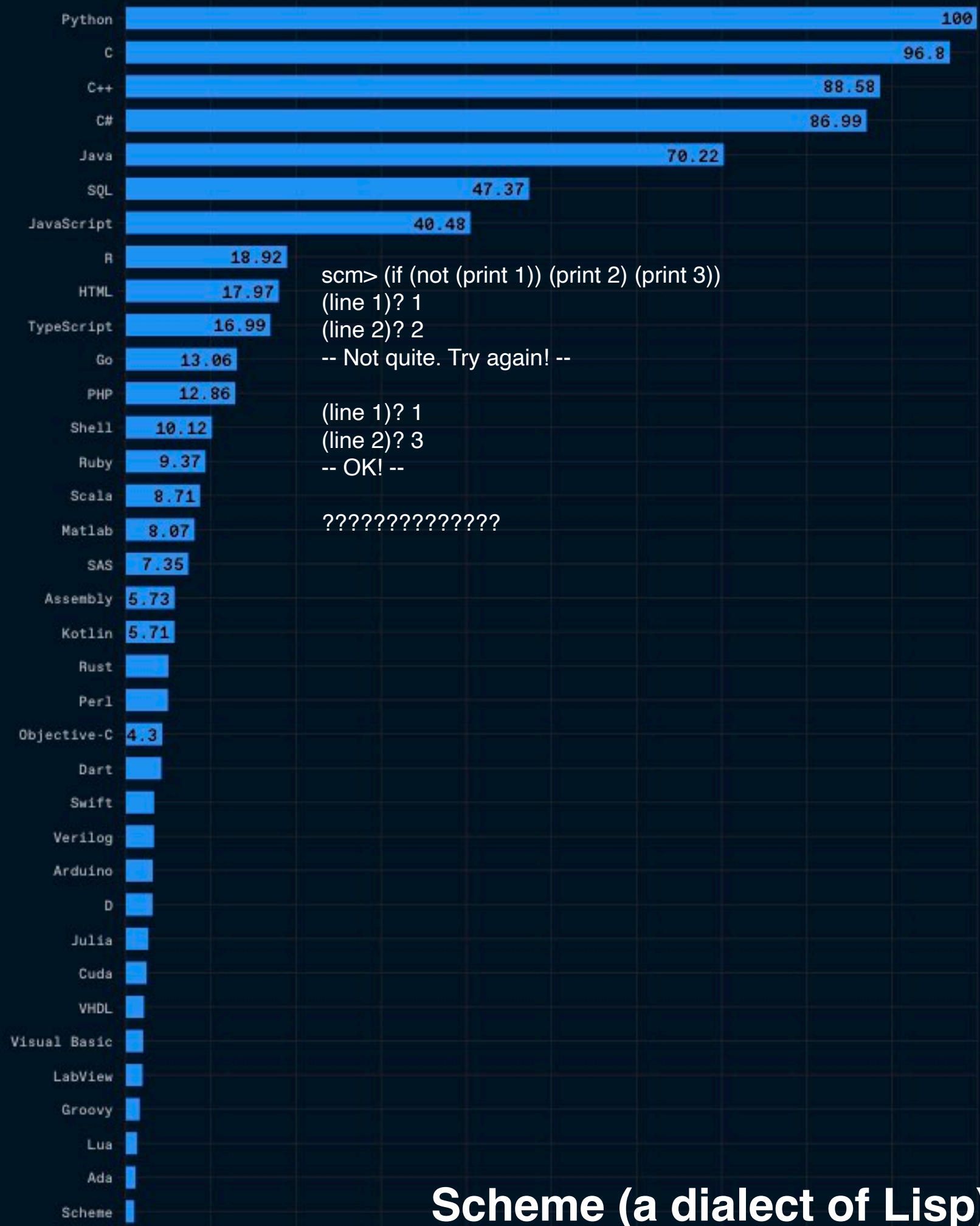
scm> (define y (+ x 4))
? y
-- OK! --

scm> y
? 7
-- OK! --

scm> (define x (lambda (y) (* y 2)))
? x
-- OK! --

scm> (x y)
? 14
-- OK! --

Programming Languages



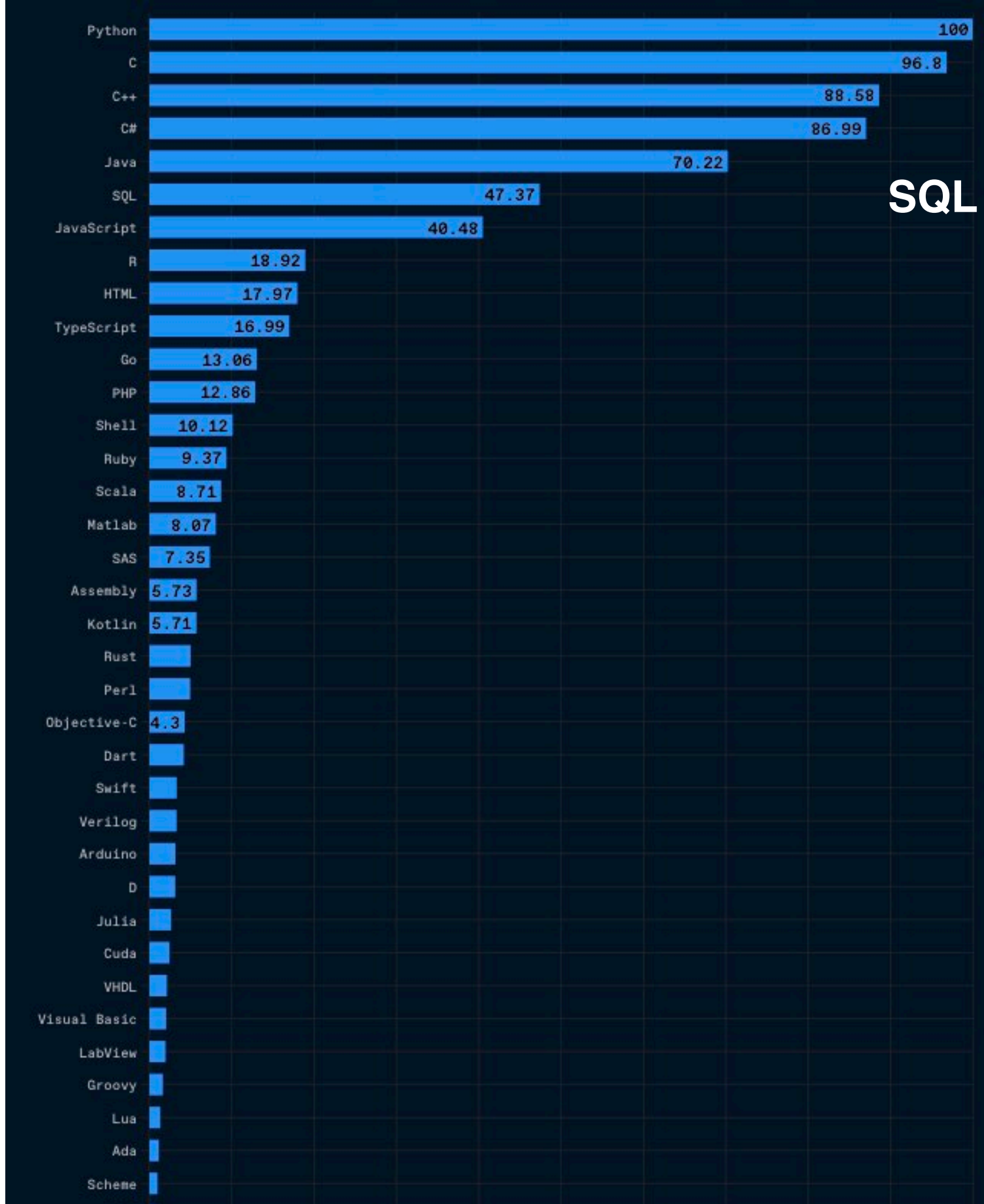
```
scm> (if (not (print 1)) (print 2) (print 3))
(line 1)? 1
(line 2)? 2
-- Not quite. Try again! --

(line 1)? 1
(line 2)? 3
-- OK! --
```

????????????????

Scheme (a dialect of Lisp)

Programming Languages



New Open Console Save Save As Share Settings Help Login

61A Code

v2.6.5

Create new file

Open existing file

Start Python interpreter

Start Scheme interpreter

Start SQL interpreter



No recent local files.

OKPy Backups

Login to view backups

Scheme Expressions

Python	Scheme
<div>1 + 2</div> <div>3 + 4 + 5 + 6</div>	<div>(+ 1 2)</div> <div>(+ 3 4 5 6)</div>

Scheme Expressions

Python	Scheme
<pre>1 + 2 3 + 4 + 5 + 6</pre>	<pre>(+ 1 2) (+ 3 4 5 6)</pre>
<pre>(2 * 4) + (3 + 5)</pre>	<pre>(+ (* 2 4) (+ 3 5))</pre>

Scheme Expressions

Python	Scheme
<pre>1 + 2 3 + 4 + 5 + 6</pre>	<pre>(+ 1 2) (+ 3 4 5 6)</pre>
<pre>(2 * 4) + (3 + 5)</pre>	<pre>(+ (* 2 4) (+ 3 5))</pre>
<pre>"string" 'string'</pre>	<pre>"string"</pre>

Scheme Expressions

```
untitled.scm (Output) ×
CS61A Scheme Web Interpreter
-----
Welcome to the 61A Scheme web interpreter!
The source for this interpreter is restricted, but you'll build it yourself as your Scheme Project!

To visualize a list, call (draw <list>).
To draw list visualizations automatically, call (autodraw).
To view an environment diagram of your entire program, call (visualize).
To launch an editor associated with your console, call (editor).
To run a doctest, call (expect <expr> <output>).

scm> (+ 1 2)
3
scm>
```


Scheme Special Forms

Python	Scheme
x = 3	(define x 3.14)

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3.14)</pre>
<pre>if x < 0: print(-x) else: print(x)</pre>	<pre>(if (< x 0) (- x) x)</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3.14)</pre>
<pre>if x < 0: print(-x) else: print(x)</pre>	<pre>(if (< x 0) (- x) x) (if (< x 0) (print (- x)) (print x))</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3.14)</pre>
<pre>if x < 0: print(-x) else: print(x)</pre>	<pre>(if (< x 0) (- x) x) (if (< x 0) (print (- x)) (print x)) (print (if (< x 0) (- x) x))</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3.14)</pre>
<pre>if x < 0: print(-x) else: print(x)</pre>	<pre>(if (< x 0) (- x) x) (if (< x 0) (print (- x)) (print x)) (print (if (< x 0) (- x) x))</pre>
<pre>def abs(x): if x < 0: return -x else: return x</pre>	<pre>(define (abs x) (if (< x 0) (- x) x))</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3.14)</pre>
<pre>if x < 0: print(-x) else: print(x)</pre>	<pre>(if (< x 0) (- x) x) (if (< x 0) (print (- x)) (print x)) (print (if (< x 0) (- x) x))</pre>
<pre>def abs(x): if x < 0: return -x else: return x</pre>	<pre>(define (abs x) (if (< x 0) (- x) x))</pre> <p>unlike python, white space doesn't matter</p>

Scheme Special Forms

Python

Scheme

```
x = -1
if x < 0:
    print(x)
    print("done")
```

```
(define x -1)
(if (< x 0)
    (begin (print x)
            (print "done"))
    ))
```

Scheme Special Forms

Python	Scheme
<pre>if(x < 0): print(-1) elif(x > 0): print(1) else: print(0)</pre>	<pre>(cond ((< x 0) -1) ((> x 0) 1) (else 0))</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3 (x > 1) and (x < 10) true</pre>	<pre>(define x 3) (and (> x 1) (< x 10)) #t</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3)</pre>
<pre>(x > 1) and (x < 10)</pre>	<pre>(and (> x 1) (< x 10))</pre>
<pre>true</pre>	<pre>#t</pre>
	<pre>(or (= x 3) (< (/ x 0) 1))</pre>
	<pre>#t</pre>

Scheme Special Forms

Python	Scheme
<pre>x = 3</pre>	<pre>(define x 3)</pre>
<pre>(x > 1) and (x < 10)</pre>	<pre>(and (> x 1) (< x 10))</pre>
<pre>true</pre>	<pre>#t</pre>
	<pre>(or (= x 3) (< (/ x 0) 1))</pre>
	<pre>#t</pre>
	<pre>(define x 2)</pre>
	<pre>(or (= x 3) (< (/ x 0) 1))</pre>
	<pre>Error: division by zero</pre>

Scheme Lambda Expressions

Python	Scheme
<pre>f = lambda x,y,z: x+y+abs(z) f(1,2,3)</pre>	<pre>(define f (lambda (x y z) (+ x y (abs z)))) (f 1 2 3)</pre>

Scheme Let Expressions

Python	Scheme
<pre>a = 3 b = 4 c = math.sqrt(a*a + b*b) # a and b remain bound</pre>	<pre>(define c (let ((a 3) (b 4)) (sqrt (+ (* a a) (* b b))))) ; a and b are not bound here</pre>

Scheme Let Expressions

Python	Scheme
	<pre>; inline comment ;; short comment on own line ;;; longer explanatory comment</pre>

Scheme

```
(number? 3)
```

```
#t
```

```
(zero? 2)
```

```
#f
```

```
(integer? 2.2)
```

```
#f
```

```
(equal? 2 2) ; see also eq?
```

```
#t
```

```
(string? "foo")
```

```
#t
```

Python	Scheme
<pre>for x in range(10): print(x) x = 0 while(x < 10): print(x) x = x + 1</pre>	

Python

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)
```

Scheme

Python	<pre>def factorial(n): if n == 0: return 1 else: return n * factorial(n-1)</pre>
Scheme	<pre>(define (factorial n) (if (= n 0) 1 (* n (factorial (- n 1)))))</pre>

Python	<pre>def perfectSquare(x,i=0): if i > sqrt(x): return False else: return i*i==x or perfectSquare(x,i+1)</pre>
Scheme	

Python

```
def perfectSquare(x,i=0):  
    if i > sqrt(x):  
        return False  
    else:  
        return i*i==x or perfectSquare(x,i+1)
```

Scheme

```
(define (perfectSquare n i)  
  (if (> i (sqrt n))  
      #f  
      (or (= (* i i) n) (perfectSquare n (+ i 1)))))  
(perfectSquare 16 0)
```

Python	<pre>def perfectSquare(x,i=0): if i > sqrt(x): return False else: return i*i==x or perfectSquare(x,i+1)</pre>
Scheme	<pre>(define (perfectSquare n i) (if (> i (sqrt n)) #f (or (= (* i i) n) (perfectSquare n (+ i 1))))) (perfectSquare 16 0)</pre>
Scheme	<pre>(define (perfectSquare n) (define (psHelper n i) (if (> i (sqrt n)) #f (or (= (* i i) n) (psHelper n (+ i 1))))) (psHelper n 0)) (perfectSquare 16)</pre>

Python

```
def fib(n):  
    if n == 1:  
        return 0  
    elif n == 2:  
        return 1  
    else:  
        return fib(n-2) + fib(n-1)
```

Scheme

Python

```
def fib(n):  
    if n == 1:  
        return 0  
    elif n == 2:  
        return 1  
    else:  
        return fib(n-2) + fib(n-1)
```

Scheme

```
(define (fib n)  
  (cond  
    ((= n 0) 0)  
    ((= n 1) 1)  
    (else (+ (fib (- n 2)) (fib (- n 1)))))  
  ))
```

Python

```
def countNines(num):  
    if( num == 0 ):  
        return 0  
    else:  
        if( num % 10 == 9 ):  
            return 1 + countNines(num//10)  
        else:  
            return countNines(num//10)
```

Scheme

Python

```
def countNines(num):  
    if( num == 0 ):  
        return 0  
    else:  
        if( num % 10 == 9 ):  
            return 1 + countNines(num//10)  
        else:  
            return countNines(num//10)
```

Scheme

```
(define (countNines num)  
  (if (= num 0)  
      0  
      (if (= (modulo num 10) 9 )  
          (+ 1 (countNines (floor (/ num 10))))  
          (countNines (floor (/ num 10))))  
      )))
```

Python

```
import math
def isPrime(n,i=2):
    if i > math.sqrt(n):
        return True
    else:
        if n % i == 0:
            return False
        else:
            return isPrime(n,i+1)
isPrime(9)
```

Scheme

Python

```
import math
def isPrime(n,i=2):
    if i > math.sqrt(n):
        return True
    else:
        if n % i == 0:
            return False
        else:
            return isPrime(n,i+1)
isPrime(9)
```

Scheme

```
(define (isPrime n i)
  (if (> i (sqrt n))
      #t
      (if (= (modulo n 2) 0)
          #f
          (isPrime n (+ i 1)))))
(isPrime 9 2)
```

Python

```
import math
def isPrime(n,i=2):
    if i > math.sqrt(n):
        return True
    else:
        if n % i == 0:
            return False
        else:
            return isPrime(n,i+1)
isPrime(9)
```

Scheme

```
(define (isPrime n i)
  (cond ((> i (sqrt n)) #t)
        ((= (modulo n 2) 0) #f)
        (else (isPrime n (+ i 1))))
  ))
(isPrime 9 2)
```

```
(define (single-digit x)  
  ; return true if x >= 0 and x < 10)
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(if (= 5 (+ 2 3)) 10 20)
```

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(if (= 5 (+ 2 3)) 10 20)
```

```
10
```



```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(if (= 5 (+ 2 3)) 10 20)
```

```
10
```

```
(define (my-max x y)
```

```
  ; your code)
```

```
(my-max 20 10)
```

```
20
```

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(if (= 5 (+ 2 3)) 10 20)
```

```
10
```

```
(define (my-max x y)
```

```
  (if (> x y) x y))
```

```
(my-max 20 10)
```

```
20
```

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

#t

```
(single-digit 12)
```

#f

```
(if (= 5 (+ 2 3)) 10 20)
```

10

```
(define (my-max x y)
```

```
  (if (> x y) x y))
```

```
(my-max 20 10)
```

20

```
(define (my-max3 x y z)
```

```
  (if (and ???)
```

x

```
    (if ???
```

y

```
      z)))
```

```
(my-max3 4 12 8)
```

12

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(if (= 5 (+ 2 3)) 10 20)
```

```
10
```

```
(define (my-max x y)
```

```
  (if (> x y) x y))
```

```
(my-max 20 10)
```

```
20
```

```
(define (my-max3 x y z)
```

```
  (if (and (> x y) (> x z))
```

```
      x
```

```
      (if (> y z)
```

```
          y
```

```
          z))))
```

```
(my-max3 4 12 8)
```

```
12
```

```
(define (single-digit x)
  (and (>= x 0) (< x 10)))
```

```
(single-digit 5)
```

```
#t
```

```
(single-digit 12)
```

```
#f
```

```
(if (= 5 (+ 2 3)) 10 20)
```

```
10
```

```
(define (my-max x y)
```

```
  (if (> x y) x y))
```

```
(my-max 20 10)
```

```
20
```

```
(define (my-max3 x y z)
  (if (and (> x y) (> x z))
```

```
    x
```

```
    (if (> y z)
```

```
        y
```

```
        z))))
```

```
(my-max3 4 12 8)
```

```
12
```

```
(define (my-max3 x y z)
  (cond ((and (> x y) (> x z)) x)
        ((> y z) y)
        (else z)))
```