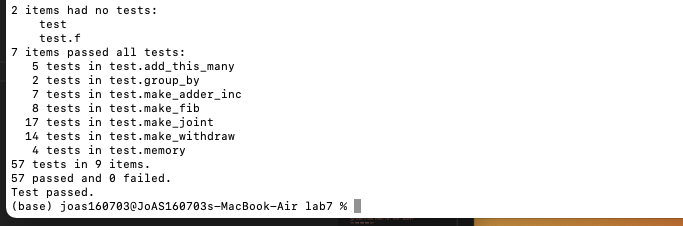
### **CIS 61 :: Lab 06 - Mutable Sequences and Functions**



Q1 - Add This Many

Write a function that takes in a value x, a value el, and a list and adds as many el’s to the end of the list as there are x’s. Make sure to modify the original list using list mutation techniques.

|  |
| --- |
| def add\_this\_many(x, el, lst):  """ Adds el to the end of lst the number of times x occurs in lst.  >>> lst = [1, 2, 4, 2, 1]  >>> add\_this\_many(1, 5, lst)  >>> lst  [1, 2, 4, 2, 1, 5, 5]  >>> add\_this\_many(2, 2, lst)  >>> lst [1, 2, 4, 2, 1, 5, 5, 2, 2]  """ |

Q2 - Group By

Write a function that takes in a sequence s and a function fn and returns a dictionary.

The values of the dictionary are lists of elements from s. Each element e in a list should be constructed such that fn(e) is the same for all elements in that list.

Finally, the key for each value should be fn(e).

|  |
| --- |
| def group\_by(s, fn):  """  >>> group\_by([12, 23, 14, 45], lambda p: p // 10)  {1: [12, 14], 2: [23], 4: [45]}  >>> group\_by(range(-3, 4), lambda x: x \* x)  {0: [0], 1: [-1, 1], 4: [-2, 2], 9: [-3, 3]}  """ |

### 

### Q3: Make Adder Increasing

Write a function which takes in an integer n and returns a one-argument function. This function should take in some value x and return n + x the first time it is called, similar to make\_adder. The second time it is called, however, it should return n + x + 1, then n + x + 2 the third time, and so on.

|  |
| --- |
| **def** **make\_adder\_inc**(n):  """  >>> adder1 = make\_adder\_inc(5)  >>> adder2 = make\_adder\_inc(6)  >>> adder1(2)  7  >>> adder1(2) # 5 + 2 + 1  8  >>> adder1(10) # 5 + 10 + 2  17  >>> [adder1(x) for x in [1, 2, 3]]  [9, 11, 13]  >>> adder2(5)  11  """  "\*\*\* YOUR CODE HERE \*\*\*" |

Nonlocal

Q5 - Memory

Write a function that takes in a number n and returns a one-argument function. The returned function takes in a function that is used to update n. It should return the updated n.

|  |
| --- |
| def memory(n):  """  >>> f = memory(10)  >>> f(lambda x: x \* 2)  20  >>> f(lambda x: x - 7)  13  >>> f(lambda x: x > 5)  True  """ |

### Q6: Next Fibonacci

Write a function make\_fib that returns a function that returns the next Fibonacci number each time it is called. (The Fibonacci sequence begins with 0 and 1, after which each element is the sum of the preceding two.) Use a nonlocal statement!

|  |
| --- |
| **def** **make\_fib**():  """Returns a function that returns the next Fibonacci number  every time it is called.  >>> fib = make\_fib()  >>> fib()  0  >>> fib()  1  >>> fib()  1  >>> fib()  2  >>> fib()  3  >>> fib2 = make\_fib()  >>> fib() + sum([fib2() for \_ in range(5)])  12  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q7: Password Protected Account

In lecture, we saw how to use functions to create mutable objects. Here, for example, is the function make\_withdraw which produces a function that can withdraw money from an account:

|  |
| --- |
| def make\_withdraw(balance):  """Return a withdraw function with BALANCE as its starting balance.  >>> withdraw = make\_withdraw(1000)  >>> withdraw(100)  900  >>> withdraw(100)  800  >>> withdraw(900)  'Insufficient funds'  """  def withdraw(amount):  nonlocal balance  if amount > balance:  return 'Insufficient funds'  balance = balance - amount  return balance  return withdraw |

Write a version of the make\_withdraw function that returns password-protected withdraw functions. That is, make\_withdraw should take a password argument (a string) in addition to an initial balance. The returned function should take two arguments: an amount to withdraw and a password.

A password-protected withdraw function should only process withdrawals that include a password that matches the original. Upon receiving an incorrect password, the function should:

1. Store that incorrect password in a list, and
2. Return the string 'Incorrect password'.

If a withdraw function has been called three times with incorrect passwords <p1>, <p2>, and <p3>, then it is locked. All subsequent calls to the function should return:

"Your account is locked. Attempts: [<p1>, <p2>, <p3>]"

The incorrect passwords may be the same or different:

|  |
| --- |
| def make\_withdraw(balance, password):  """Return a password-protected withdraw function.  >>> w = make\_withdraw(100, 'hax0r')  >>> w(25, 'hax0r')  75  >>> error = w(90, 'hax0r')  >>> error  'Insufficient funds'  >>> error = w(25, 'hwat')  >>> error  'Incorrect password'  >>> new\_bal = w(25, 'hax0r')  >>> new\_bal  50  >>> w(75, 'a')  'Incorrect password'  >>> w(10, 'hax0r')  40  >>> w(20, 'n00b')  'Incorrect password'  >>> w(10, 'hax0r')  "Your account is locked. Attempts: ['hwat', 'a', 'n00b']"  >>> w(10, 'l33t')  "Your account is locked. Attempts: ['hwat', 'a', 'n00b']"  >>> type(w(10, 'l33t')) == str  True  """  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q8: Joint Account

Suppose that our banking system requires the ability to make joint accounts. Define a function make\_joint that takes three arguments.

1. A password-protected withdraw function,
2. The password with which that withdraw function was defined, and
3. A new password that can also access the original account.

The make\_joint function returns a withdraw function that provides additional access to the original account using *either* the new or old password. Both functions draw from the same balance. Incorrect passwords provided to either function will be stored and cause the functions to be locked after three wrong attempts.

*Hint*: The solution is short (less than 10 lines) and contains no string literals! The key is to call withdraw with the right password and amount, then interpret the result. You may assume that all failed attempts to withdraw will return some string (for incorrect passwords, locked accounts, or insufficient funds), while successful withdrawals will return a number.

Use type(value) == str to test if some value is a string:

|  |
| --- |
| def make\_joint(withdraw, old\_password, new\_password):  """Return a password-protected withdraw function that has joint access to the balance of withdraw.  >>> w = make\_withdraw(100, 'hax0r')  >>> w(25, 'hax0r')  75  >>> make\_joint(w, 'my', 'secret')  'Incorrect password'  >>> j = make\_joint(w, 'hax0r', 'secret')  >>> w(25, 'secret')  'Incorrect password'  >>> j(25, 'secret')  50  >>> j(25, 'hax0r')  25  >>> j(100, 'secret')  'Insufficient funds'  >>> j2 = make\_joint(j, 'secret', 'code')  >>> j2(5, 'code')  20  >>> j2(5, 'secret')  15  >>> j2(5, 'hax0r')  10  >>> j2(25, 'password')  'Incorrect password'  >>> j2(5, 'secret')  "Your account is locked. Attempts: ['my', 'secret', 'password']"  >>> j(5, 'secret')  "Your account is locked. Attempts: ['my', 'secret', 'password']"  >>> w(5, 'hax0r')  "Your account is locked. Attempts: ['my', 'secret', 'password']"  >>> make\_joint(w, 'hax0r', 'hello')  "Your account is locked. Attempts: ['my', 'secret', 'password']"  """  "\*\*\* YOUR CODE HERE \*\*\*" |