

CS10 Paper Final – Summer 2018

Your Name (first, last)

ID Card Number

Your TA's Name

← _____
Name of person on left (or aisle)

_____ →
Name of person on right (or aisle)

Fill in the correct circles & squares completely...like this: ● (select ONE) ■ (select ALL that apply)

There are 85 points total for this exam and you have 180 minutes to complete this exam. Use your time wisely.

You get 1 point for putting your ID card number on each page.

Questions 1 – 12: What's That Smell? It's Potpourri! (32 points total; 35 min. recommended)

1) (2 pts) Currently, computer scientists develop machines with artificial intelligence to act as rational agents. This means that they are *always* trying to maximize:

- ☐ Human accessibility
- ☒ Expected utility
- ☐ Autonomy
- ☐ Efficiency
- ☐ Accuracy

2) (3 pts) You want to speed up your final project by using an improved processor. So you read through your code and realize that 20% of it is parallelizable. Using Intel's new 64-core processor, you can achieve a maximum speedup of 1.245 times (or 24.5%). What is the maximum speedup you could attain using an infinite number of cores?

1.25 times (25%)

(Hint: Remember that the quotient of any number divided by infinity can be approximated as zero.)

3) (2 pts) If a data type in Python is an *iterable*, what are you always allowed to do with objects of that data type?

- ☐ Find a key in the object
- ☒ Move through the data contained in the object with a for loop
- ☐ Add and remove items from the object
- ☐ Write a comment attached to the object
- ☐ Convert the object into a different data type

ID Card Number: _____

4) (2 pts) Based on the Algorithmic Bias lecture, why does Google Translate convert the gender-neutral Turkish phrase “o bir doktor” into the gendered English phrase “he is a doctor”?

- ☐ English has no gender-neutral pronouns, so the algorithm is forced to use the word “he.”
- ☐ In English speaking countries, most doctors are men, so the algorithm uses the word “he.”
- ☐ The algorithm randomly chooses a gendered pronoun when translating gender-neutral words into English.
- ☒ The algorithm was trained using a dataset in which the word “doctor” was frequently associated with male pronouns.
- ☐ The algorithm actively tries to reproduce gender biases that already exist in the English language.

5) (2 pts) True or False: If you found an efficient solution to the knapsack problem, you could find an efficient solution to password decryption.

- ☒ True
- ☐ False

6) (2 pts) As described in his guest lecture on Natural Language Processing, why did Nick Adams create the online text-parsing tool “TagWorks”?

- ☐ He was displeased with existing interfaces for parsing textual documents.
- ☒ He wanted to create a tool to allow humans to help computers parse text, because computational text parsing is still at a very primitive stage of development.
- ☐ He thought that existing parsing tools were too reliant on humans, and wanted to create a tool that was entirely computer-driven.
- ☐ He realized that all existing NLP tools were only calibrated to handle numerical data, and needed a tool that could handle textual data.
- ☐ None of the above.

7) (3 pts) Quinary is a number system that utilizes base-5, the same way the decimal number system utilizes base-10. Convert the decimal number 128 into quinary. For your reference, $5^2 = 25$, $5^3 = 125$, and $5^4 = 625$.

1003

8) (2 pts) Which of the following exciting and/or problematic issues were mentioned in Schuyler’s lecture on the Future of Computing? **Choose all that apply.**

- ☒ Quantum computing
- ☒ Income inequality and housing in the Bay Area
- ☐ Biological computing and DNA-based data
- ☒ Self-driving cars and automation
- ☐ The singularity and extreme AI

9) (4 pts) Rank the following algorithms from slowest to fastest.

- Finding a number in an unsorted list by looking at each number one-by-one.
- Finding a number in a sorted list by comparing the number to the number in the center and discarding half of the list repeatedly.
- Finding a number in an unsorted list by looking at an infinite list of memory where all of the positions of the numbers have been saved.
- Finding a number in an unsorted list by:
 - first sorting the list by comparing each number against every other number in the list
 - then running a binary search algorithm on the list.

Slowest D A B C Fastest

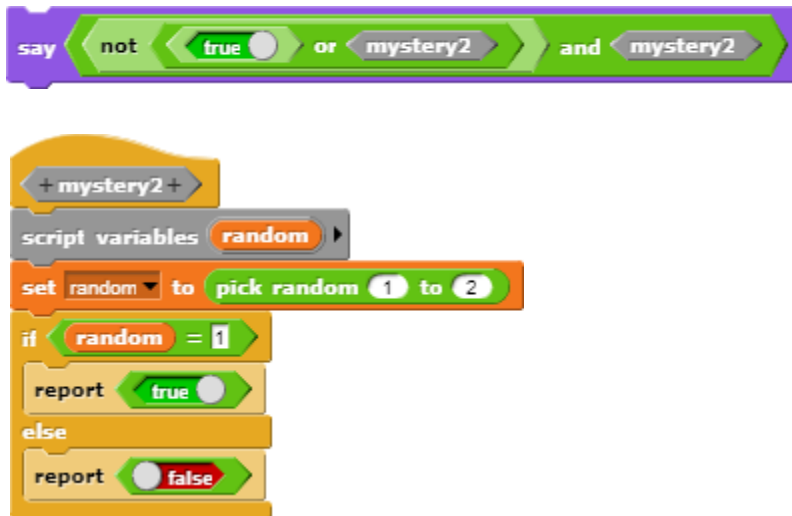
10) (4 pts total) For each of the following code snippets, write what the Sprite would say after the script executes. If you believe the code causes an infinite loop (i.e., runs forever), write "Loop." If you believe the code produces any other error message, write "Error."

a) (2 pts)



10

b) (2 pts)



False

ID Card Number: _____

11) (3 pts) We are trying to write a Python function called `repeat_seven`, which takes as input a list, and returns a new list containing the input list repeated seven times. For example:

```
>>> my_list = [1, 2]
>>> repeat_seven(my_list)
[[1, 2], [1, 2], [1, 2], [1, 2], [1, 2], [1, 2], [1, 2]]
```

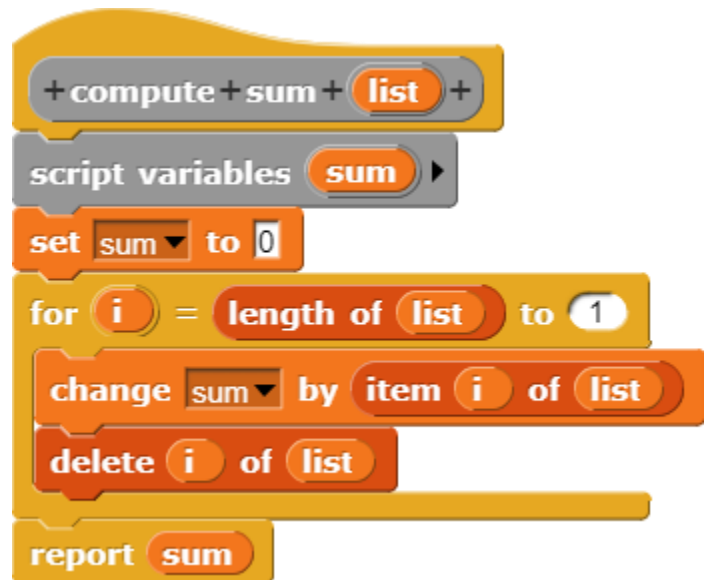
Complete our definition of `repeat_seven` below by writing a one-line list comprehension. Solutions that are more than one line or do not use a list comprehension will not receive credit.

```
def repeat_seven(input_list):
    return [input_list for i in range(7)]
```

12) (3 pts) Why is the code to the right a terrible example of a function? **Write a brief explanation in the box provided below.**

(Hint: Think about the definition of a function given in Jobel's lecture on "Functions & Procedures." Is the code to the right a function? Should it be?)

This block causes a side-effect: mutating the original input list. The input list will always be converted into an empty list. According to Jobel's lecture on functions, a function should never cause side-effects. What's worse, this block is a reporter, so users will expect it to simply report an output, without mutating the input list.




Question 13: Extra Credit (1 point)

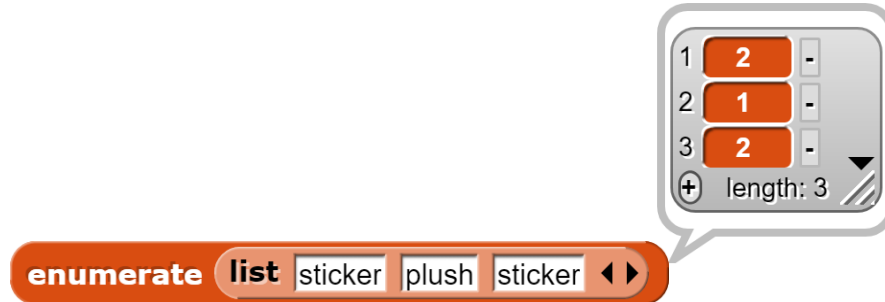
13) What is one thing you learned during the Alumni Panel? (Don't make something up...we were all there, and we know what was said.)

There were many possible correct answers to question #13, far too many to list here.

Question 14: Alonzo Anonymous (8 points total, 25 min.)

Alonzo has a store of new CS10 merchandise to show off, but needs to take inventory before they can be put on sale. Inside the store's computer systems, every individual item is stored in a warehouse inventory list.

We need to write a block called **enumerate**  that will tell us the number of times each item appears in the store's inventory. We want this block to take in the inventory list and return a *new list* where each item is replaced by the number of times it appears in the input list. Take the following example:



a) (3 pts) We've come up with a few different ways of solving this question, but Alonzo isn't sure which one is correct. Help Alonzo out by choosing the correct implementation.





☐ **map** **keep items such that**  **contains** **#1** **from** **inventory**
input names: **#1** **over** **inventory**

☐ **length of** **map** **keep items such that**  **#1** **=**  **from** **inventory** **over** **inventory**

☒ **map** **length of** **keep items such that**  **#1** **=**  **from** **inventory**
input names: **#1** **over** **inventory**



☐ **map** **length of** **keep items such that** **inventory** **contains** **#1** **from** **inventory**
input names: **#1** **over** **inventory**

ID Card Number: _____

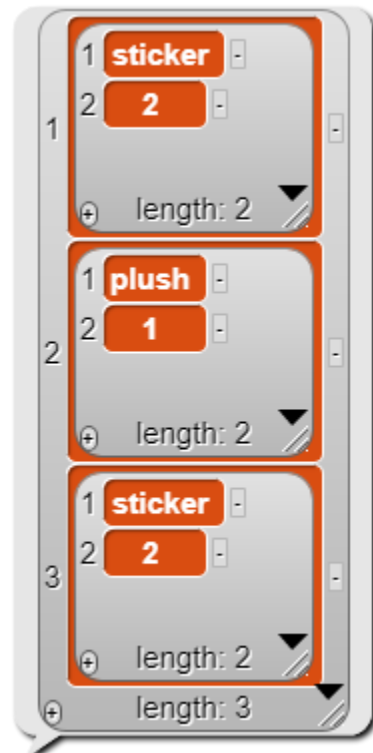
b) (2 pts) Assuming that the , , and  blocks all execute in constant time, what is the worst case runtime of  as a function of the size of the input list?

<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Constant	Logarithmic	Linear	Quadratic	Cubic	Exponential

(Hint: Don't worry about whether your answer to part (a) was correct. All four of the possible implementations listed have the same runtime (given the assumptions above).)

c) (3 pts) Now, using the  block, write the  block, which takes in an inventory list and returns a list of two-item sublists. The first item in each sublist is a value in the input list (let's call it X), and the second item is the number of times the value X appears in the input list. For example:



 list sticker plush sticker

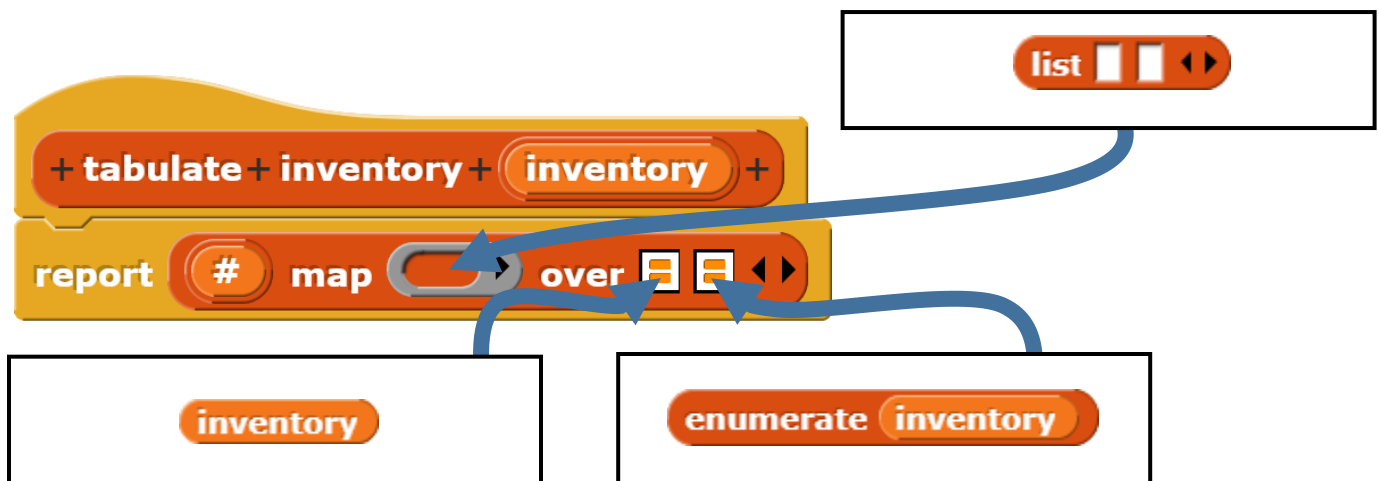


To do this, we're going to need to use the multi-list map. This map takes in two lists and a function with two blank input slots, and applies the function to the items of both lists in a sequential, pairwise fashion. You used this version of map when completing HW2. Below is an example call:

 # map + over list 1 2 list 3 4



Fill in the blanks to complete the following implementation of . You do not need to remove duplicates from the input list. Assume you have a working  block.



For questions 15 & 16, you may find it useful to reference the implementation of Count Change we built during lecture. A copy is provided on the sheet at the end of this exam.

Question 15: Count Change It Up (9 points total, 25 min.)

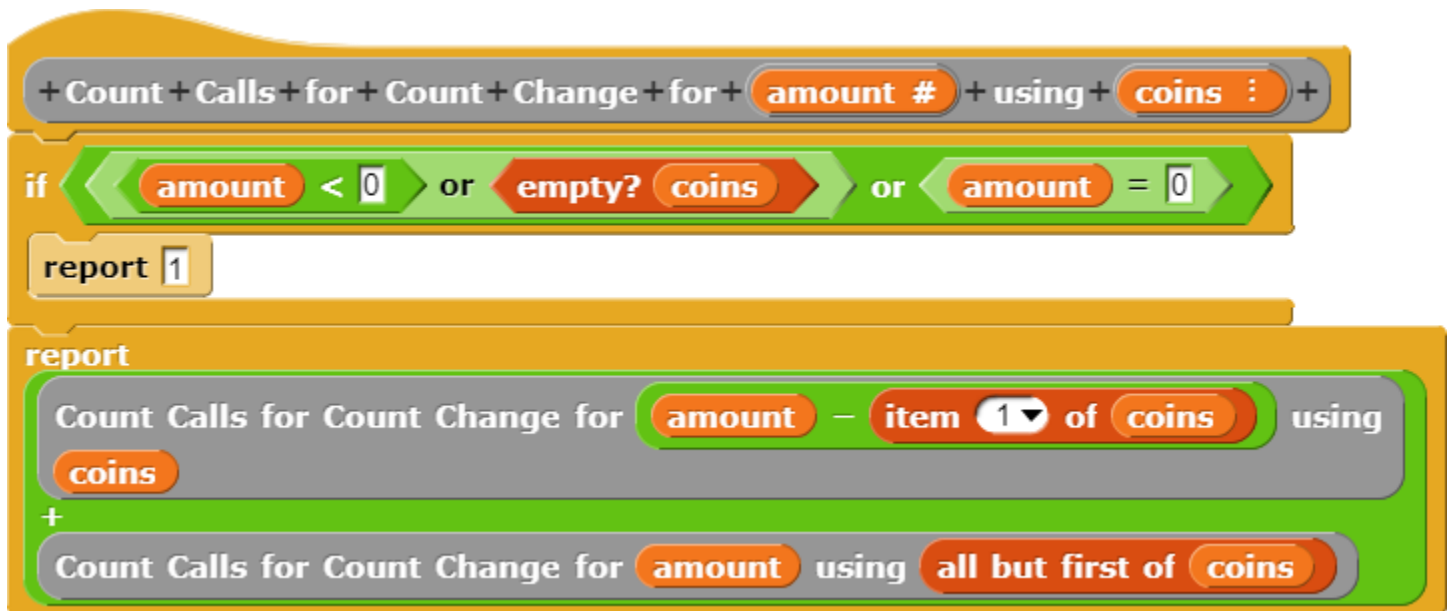
As we're sure you'll agree, Count Change is a pretty great function. But unfortunately its runtime is terribly inefficient. To demonstrate this, we've tried to write the block "Count Calls for Count Change." It takes as input an amount (in cents) and a list of coins, and should output the number of calls (recursive or non-recursive) to Count Change required to compute a result for these inputs. For some example calls, see the table below:

Function Call	Explanation
Count Calls for Count Change for 0 using list 25 10 5 1 13927	Since counting change for amount=0 is a base case, this only requires one call to Count Change (the initial call).
Count Calls for Count Change for 1 using list 1 3	This requires three total calls: the initial call, and recursive call with amount=0 and coins = [1], and the recursive call with amount = 1 and coins = [].
Count Calls for Count Change for 100 using list 25 10 5 1 13927	Yeah, so...it's inefficient.

Question continued on next page....

ID Card Number: _____

Below is our attempt to implement Count Calls for Count Change. Unfortunately, though, it has a bug.



a) (3 pts) Ideally, if Count Calls for Count Change is working properly, what should the call below report? **Write your answer in the box provided.**

Count Calls for Count Change for 5 using list 5 1

13

b) (3 pts) Using the buggy implementation above, what will the call below report? **Write your answer in the box provided.**

Count Calls for Count Change for 5 using list 5 1

7

c) (3 pts) Below, describe how you can modify our buggy version of Count Calls for Change so that it works properly. **Note: It is possible to fix the code with a very simple modification. If your answer is unnecessarily long or complex, it may not receive full credit.**

Add "+ 1" to the recursive case.

Question 16: Count Change It Up Again (8 points total, 20 min.)

Another problem with Count Change is that it's very unrealistic. When will anyone ever have an unlimited supply of coins? So, let's write a more realistic version of Count Change: "Count Change with Limits." It takes as input an amount and list of coins, similar to the original Count Change. But our new function also takes a third argument, "LIMIT," which indicates the maximum number of coins we can use to make change. It returns the number of ways to make change for AMOUNT using the given COINS and LIMIT.

Count Change with Limits doesn't care which coins we use; if the limit is 3, we can use 3 nickels or 2 dimes and a penny. But the function will never use more coins than the limit allows. Again, see below for sample calls.

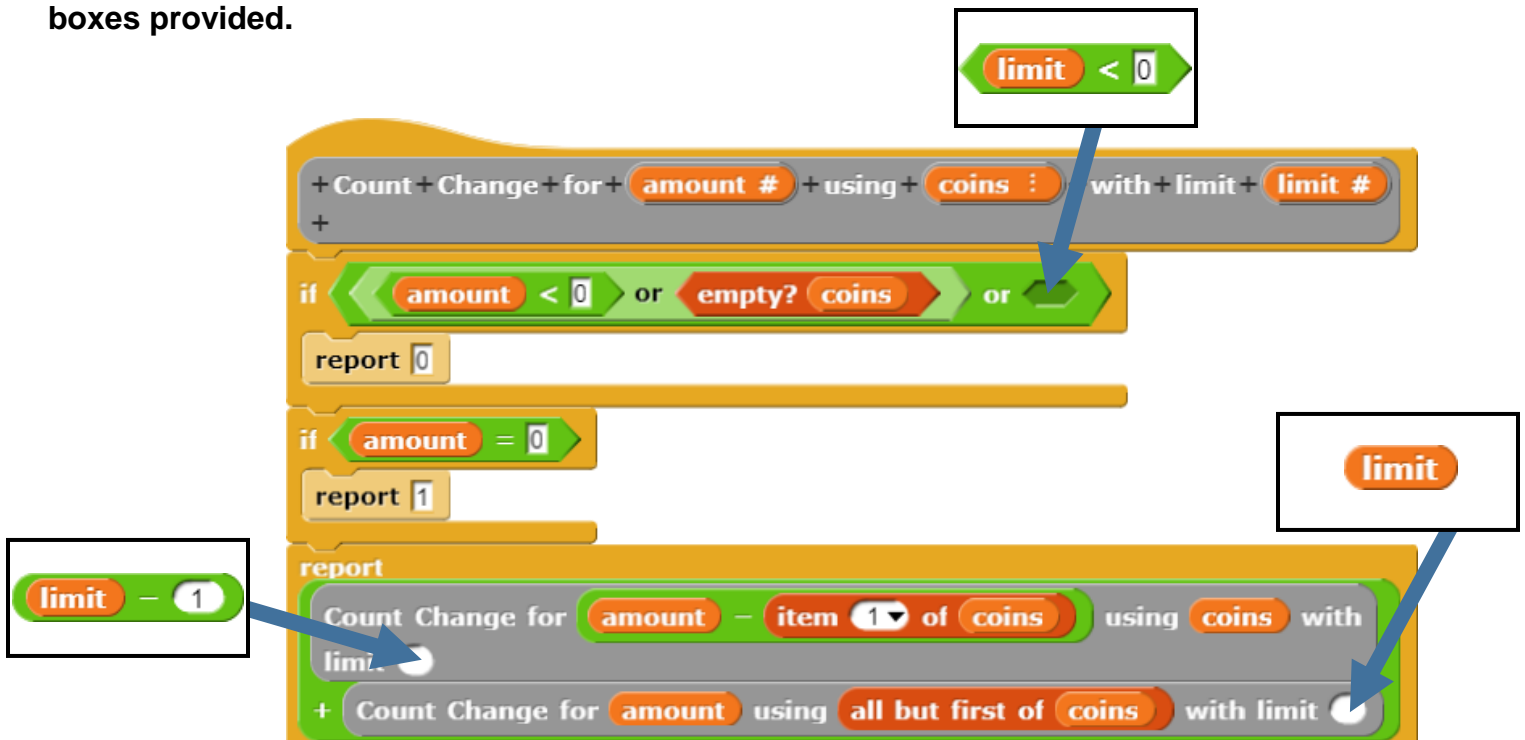


a) (2 pts) As a sanity check, what should the following call to Count Change with Limits report? **Write your answer in the box provided.**

Count Change for 25 using list 25 10 5 1 with limit 5

4

b) (6 pts) Complete our implementation of Count Change with Limits below. **Write all answers in the boxes provided.**



ID Card Number: _____

Question 17: Object-Oriented Alonzo (9 points total, 25 min.)

Assume we start up the Python interpreter and execute the code below. What will the interpreter print after each of the following commands? If you believe the code throws any sort of error message, just write "Error." If you believe nothing is printed, write "N/A." Write all answers in the boxes provided.

Note: These questions are NOT independent. You should assume that all lines of code are executed sequentially as you progress down the page.

```
class Alonzo_Generator:
```

```
    def __init__(self, number):
        self.number = number
        number += 1
```

```
    def generate_alonzo(self, name, color, age):
        alonzo = Alonzo(name, color, age)
        return alonzo
```

```
class Alonzo:
```

```
    floats = True
    enemy = "Terminalonzo"
```

```
    def __init__(self, name, color, age):
        self.name = name
        self.color = color
        self.age = 100 + age
```

```
    def feels_generator(self):
        self.intro = "I am feeling "
        return lambda x: self.intro + x
```

```
class Jobelonzo(Alonzo):
```

```
    floats = False
    age = 80
```

```
    def __init__(self, name):
        self.name = name
```

```
    def teach(self):
        print("Computing in the News")
```

```
>>> gen1 = Alonzo_Generator(1)
>>> gen1.number
```

1

```
>>> my_alonzo =
gen1.generate_alonzo("Alonzo", "yellow", 0)
>>> my_alonzo.age
```

100

```
>>> terminalonzo =
gen1.generate_alonzo("Terminalonzo", "grey", 5)
>>> terminalonzo.enemy = "Alonzo"
>>> my_alonzo.enemy
```

'Terminalonzo'

```
>>> spicelonzo =
gen1.generate_alonzo("Spicelonzo", "red", 10)
>>> spicelonzo_feels =
spicelonzo.feels_generator()
>>> spicelonzo_feels("spicy")
```

'I am feeling spicy'

```
>>> jobel = Jobelonzo("Jobel")
>>> jobel.teach()
```

'Computing in the News'

```
>>> Alonzo.enemy = "Gobo"
>>> enemies = [Jobelonzo.enemy,
terminalonzo.enemy, my_alonzo.enemy]
>>> enemies
```

['Gobo', 'Alonzo',
'Gobo']

Question 18: Small World, Big Data (6 points total, 20 min.)

Assume we open a Jupyter notebook, import the datascience and numpy modules (numpy as np), create a table named “mystery,” and execute the code below.

```
>>> mystery.column("Rating") + 10
array([15, 15, 13, 15, 18, 20])
>>> mystery.sort("Rating", descending=False).select("Sound")
```

Sound
Moo
Purr
Bark
Roar
Hoo
Woo

```
>>> mystery.where("Rating", are.equal_to(5)).drop("Rating", "Sound")
```

Animal
Cat
Dog
Lion

```
>>> mystery.where("Sound", are.containing("oo")).drop("Rating")
```

Animal	Sound
Cow	Moo
Owl	Hoo
Alonzo	Woo

```
>>> pets = ["Dog", "Cat", "Alonzo"]
>>> are_pets = np.array([animal in pets for animal in mystery.column("Animal")])
>>> mystery = mystery.with_column("Pet?", are_pets).where("Pet?",
    are.equal_to(len(range(1,4)) == 4))
```

a) (3 pts) What does the table “mystery” look like **before** we execute the code above? Answer this question by filling in the cells below.

(Hint: Because we never reassign the variable “mystery,” you can solve this problem by looking only at the code above the dashed line.)

Animal	Sound	Rating
Cat	Purr	5
Dog	Bark	5
Cow	Moo	3
Lion	Roar	5
Owl	Hoo	8
Alonzo	Woo	10

b) (3 pts) What does the table “mystery” look like **after** we execute the code above? Answer this question by filling in the cells below. You may not need all cells.

(Hint: You can solve this problem by looking only at the code below the dashed line and the table you drew in part (a). Your answer will be marked correct as long as it is consistent with the table you drew in part (a).)

Animal	Sound	Rating	Pet?
Cow	Moo	3	False
Lion	Roar	5	False
Owl	Hoo	8	False

ID Card Number: _____

Question 19: A Not-So-Simple Question (12 points total, 30 min.)

For the purposes of this question, let's define a *compound word* as a word composed exclusively of two simple words. A *simple word* is any word that cannot be broken into two smaller words. For example, "blackboard" is a compound word because it consists of exactly two simple words: "black" and "board." "Computer," on the other hand, is a simple word. It is not composed of any shorter words. "Computerbleh" is also a simple word. While it is composed of one simple word ("computer"), "bleh" is not a simple word and thus "computerbleh" is not compound.

We want to write a function that, given a word and a list of all known simple words, returns True if the word is compound, and False otherwise. For example:

```
>>> simple_words = ["basket", "base", "ball"]
>>> compound_word("basketball", simple_words)
True
>>> compound_word("ballbase", simple_words)
True
>>> compound_word("alonzo", simple_words)
False
```

Below are two attempts to implement `compound_word`. One of them works; the other is buggy.

```
def compound_word_1(word, simple_words):
    iscompound = False
    for i in range(len(word)):
        if word[:i] in simple_words:
            if word[i:] in simple_words:
                iscompound = True
                break
    return iscompound
```

```
def compound_word_2(word, simple_words):
    i = 0
    iscompound = False
    while not iscompound and i < len(word):
        if word[:i] in simple_words:
            if word[i:] in simple_words:
                iscompound = True
                i += 1
    return iscompound
```

a) (2 pts) Which of the implementations works correctly? Write your answer in the box below.

compound_word_1

b) (2 pts) Below, describe how we can modify the buggy version so that it works properly.

Note: It is possible to fix the code with a very simple modification. If your answer is unnecessarily long or complex, it may not receive full credit.

Move the "i += 1" statement back to the indentation level of the "if word[:i] in simple_words:" statement.

Now that we've got some working implementations, let's make our function more powerful. In reality, some compound words contain more than two simple words. So we'll redefine a *compound word* as a word composed exclusively of two or more simple words. We would like to modify `compound_word` so that it properly identifies longer compound words, as per the doc tests below:

```
>>> simple_words = ["wise", "clock", "counter"]
>>> compound_word("counterclockwise", simple_words)
True
>>> compound_word("counterclockclockwisecounterclock", simple_words)
True
>>> compound_word("counterclockalonzo", simple_words)
False
```

c) (8 pts) Complete our implementation of `compound_word_3` by filling in the skeleton code below. You are required to use all lines provided; there are no extra lines in this question.

```
def compound_word_3(word, simple_words):
    word1 = ''
    index = 1
    for letter in word:
        word1 += letter
        if word1 in simple_words:
            if word[index:] in simple_words or
               compound_word_3(word[index:], simple_words)
                return True
        index += 1
    return False
```

(Hint #1: Use recursion somewhere in your implementation.)

(Hint #2: If you try to slice a list using a lower bound that is greater than the index of the last element in the list, python will simply return an empty list. It will not throw an error message. See the code below for a few examples.)

```
>>> cool_list = ["this", "list", "is", "cool"]
>>> cool_list[4:]
[]
>>> cool_list[len(cool_list):]
[]
>>> cool_list[800:900]
[]
```

ID Card Number: _____

You may use this page as scratch paper.

Writing *Snap!* code on paper (supplementary)

You will be asked to write *Snap!* code on this exam, so we've developed a technique for writing it on paper. There are a few key things to notice:

- We often write variables in **UPPERCASE**.
- We change spaces between words in block names to dashes (this makes it much easier to read).
- We use indentation just as *Snap!* does, to help us understand what is "inside" the **if**, **else**, and other Control structures. E.g., here's how you could write the **DrawSquare** and **n!** blocks:

Draw-Square (LENGTH)

```
repeat (4)
  move (LENGTH) steps
  turn-right (90) degrees
```

(N) !

```
if N = 0
  report (1)
report (N * (N - 1) !)
```

- When you want to write a list of things, write them with an open parenthesis, then the first item, second item, etc (separated by spaces) and when you're done, put a closed parenthesis. If any of your items are a sentence, you have to put quotes around the sentence. So, for example, the following list of three things would be written as the equivalent 3-element-list:

■ **(life liberty "pursuit of happiness")**.

- Similarly, a nested list just shows up as a nested set of parenthesis. So the following would be written as

■ **((Love 5) (Hate 4) (The 10))**.

- If you want to pass in a function as argument, you know the function must be surrounded by a grey-border. Here are three new conventions:

- The grey border is written as *square brackets*: []
- Blanks are written as parenthesis with underscore **_** in the middle, but common blocks that are passed in to HOFs can be simplified by just their name (and not the parens and underscores)
- Return values are written as **→ value**

- So the following would be written as:

■ **Combine[() + ()] items-of (Map[() x ()] over (1 20 3 10))**

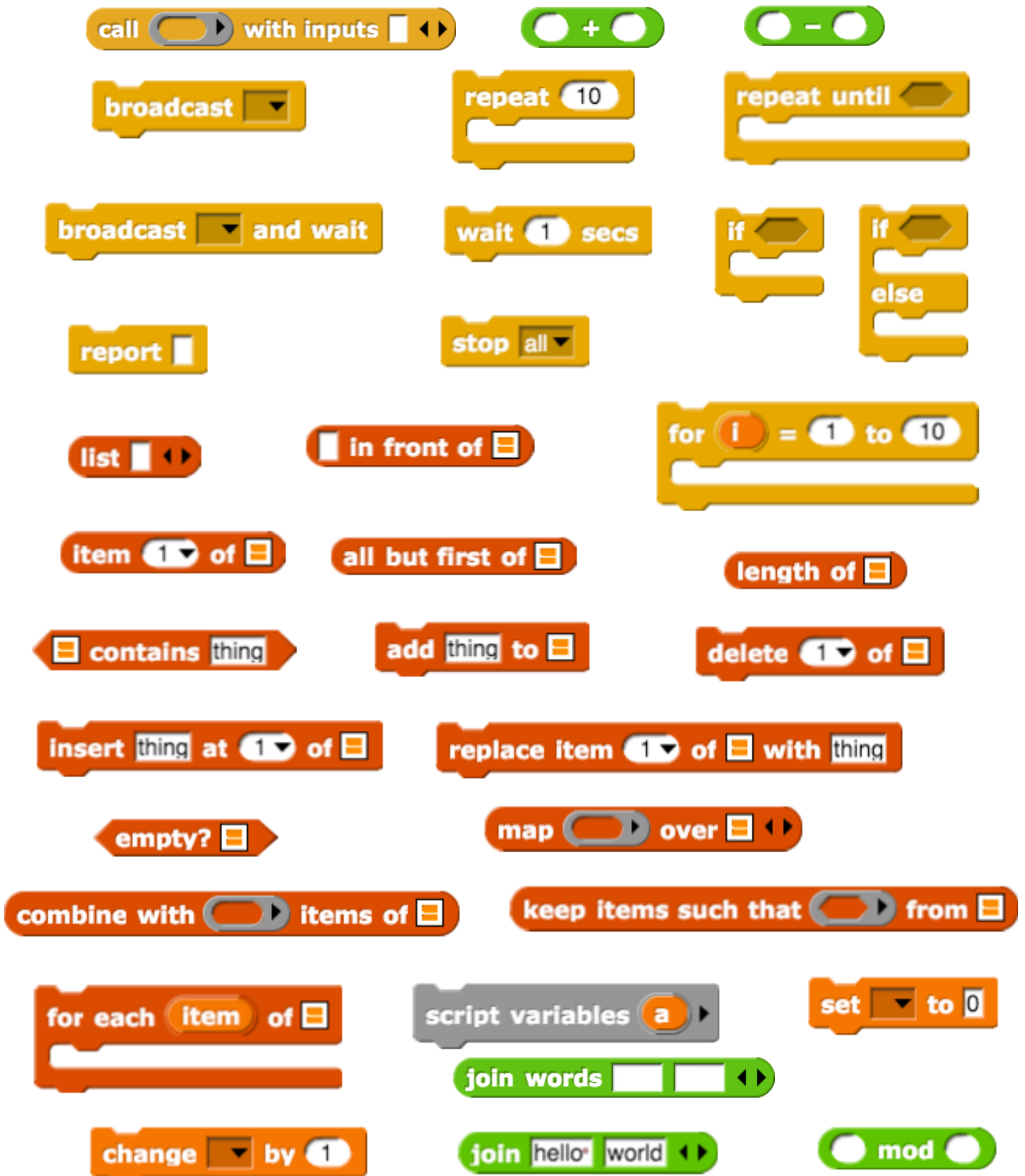
- or, in a more simplified (and preferred) format, also showing return value:

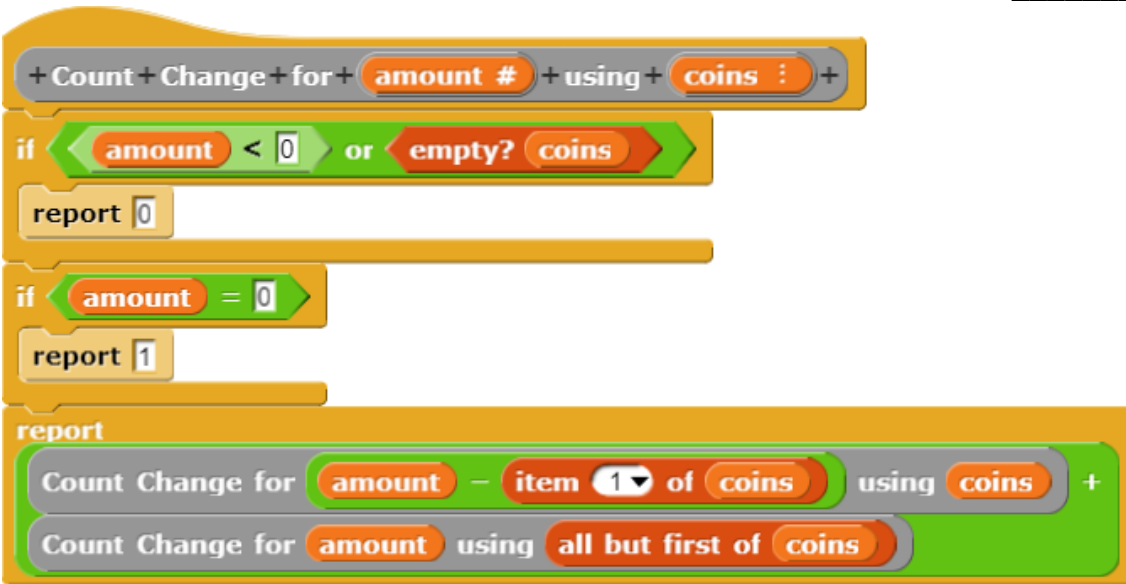
■ **Combine[+] items-of (Map[x] over (1 20 3 10)) → 51**

510

ID Card Number: _____

A bunch of Snap blocks are shown below as a reference. For coding problems on this exam, unless the problem says otherwise, you may use any Snap! block, not just the ones below (we've omitted lots of them, like x, =, split, etc.), although you do not require more than the blocks provided here. The values input in these blocks are default inputs; you may change them.





```

>>> staff = np.array(["Schuyler", "Jobel", "Jessica", "Maxson"])
>>> fav_nums = np.array([5, 87, 12, 43])
>>> fav_desserts = np.array(["Brownies", "Cookies", "Ice Cream", "Cheesecake"])
>>> staff_table = Table().with_columns(["Staff", staff, "Fav Nums", fav_nums, "Fav
    Desserts", fav_desserts])
>>> staff_table

```

Staff	Fav Nums	Fav Desserts
Schuyler	5	Brownies
Jobel	87	Cookies
Jessica	12	Ice Cream
Maxson	43	Cheesecake

```

>>> staff_table.column("Fav Desserts")

array(['brownies', 'cookies', 'ice cream', 'cheesecake'])

>>> staff_table.column("Staff").item(0)

'Schuyler'

>>> staff_table.column("Fav Nums") + 10

array([15, 97, 22, 53])

>>> staff_table.drop("Fav Nums")

```

Staff	Fav Desserts
Schuyler	Brownies
Jobel	Cookies
Jessica	Ice Cream
Maxson	Cheesecake

ID Card Number: _____

```
>>> staff_table.relabeled("Fav Desserts", "Yummy Treats")
```

Staff	Fav Nums	Yummy Treats
Schuyler	5	Brownies
Jobel	87	Cookies
Jessica	12	Ice Cream
Maxson	43	Cheesecake

```
>>> staff_table.sort("Fav Nums", descending=True)
```

Staff	Fav Nums	Fav Desserts
Jobel	87	Cookies
Maxson	43	Cheesecake
Jessica	12	Ice Cream
Schuyler	5	Brownies

```
>>> staff_table.select("Staff")
```

Staff
Schuyler
Jobel
Jessica
Maxson

```
>>> staff_table.where("Fav Nums", are.above(42))
```

Staff	Fav Nums	Fav Desserts
Jobel	87	Cookies
Maxson	43	Cheesecake

```
>>> staff_table.with_column("Likes Cookies?", staff_table.column("Fav Desserts") ==  
"Cookies").where("Staff", are.containing("J"))
```

Staff	Fav Nums	Fav Desserts	Likes Cookies?
Jobel	87	Cookies	True
Jessica	12	Ice Cream	False

Predicate	Description
<code>are.equal_to(z)</code>	Equal to <code>z</code>
<code>are.above(x)</code>	Greater than <code>x</code>
<code>are.above_or_equal_to(x)</code>	Greater than or equal to <code>x</code>
<code>are.below(x)</code>	Less than <code>x</code>
<code>are.below_or_equal_to(x)</code>	Less than or equal to <code>x</code>
<code>are.between(x, y)</code>	Greater than or equal to <code>x</code> , and less than <code>y</code>
<code>are.strictly_between(x, y)</code>	Greater than <code>x</code> and less than <code>y</code>
<code>are.between_or_equal_to(x, y)</code>	Greater than or equal to <code>x</code> , and less than or equal to <code>y</code>
<code>are.containing(s)</code>	Contains the string <code>s</code>