

CS10 Paper Midterm – Summer 2018

Alonzo, Church

011235813

Dan Garcia

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 80 points total for this exam and you have 120 minutes to complete this exam. Use your time wisely.

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

Questions 1 – 9: What's That Smell? It's Potpourri! (29 points total; 30 min. recommended)

1) (3 pts) Which of the following is NOT an example of abstraction?

- Calling a car's gas pedal an accelerator
- Removing train tracks and school locations on a map showing bus routes
- Using the Internet without understanding how the computer accesses a website
- Classifying Dog and Cat objects as part of an Animal class
- Writing a program in binary (0s and 1s) instead of Snap! or Python

2) (3 pts) True or False: Declarative programming is equally as powerful as imperative programming.

- True
- False

3) (3 pts) Which of the following could be described as a cyberattack on critical infrastructure?

- Collecting sensitive data and information by a foreign government
- Using malware to shut down power plants and sewage systems
- Hacking into a foreign university's computers
- Releasing a virus on a country's nuclear research program's computers
- Using a computer virus to gain control of self-driving cars

4) (3 pts) In Maxson Yang's lecture on computing and the environment, he described the concept of planned obsolescence. What is planned obsolescence?

- The practice of designing products that become outdated quickly and frequently
- The policy where the government decides when a product becomes outdated
- The practice of using new materials in products every few years to increase sales
- The policy of replacing devices at fixed intervals to improve efficiency
- The practice of purchasing a new device the moment a new version is released

ID Card Number: _____

5) (3 pts) Which of the following blocks can be considered “functions,” as defined in the Building Blocks lecture about functions and procedures? Assume all blocks behave correctly (i.e., without bugs). **Mark ALL answers that apply.**

- move 10 steps
- true
- delete 1 of
- join hello world
- pick random 1 to 10

6) (4 pts) Convert the hexadecimal number 0x1F3 into both binary and decimal. **Write your answers in the corresponding boxes below.** For your reference, $2^6 = 64$, $2^7 = 128$, and $2^8 = 256$. (Hint: It may be faster to convert the number into binary first, and then from binary into decimal.)

Binary 0b000111110011	Decimal 499
--------------------------	----------------

7) (3 pts) Which of the following is NOT a scientific use of computing discussed in Kathy Yelick’s lecture?

- Mitigating global climate change
- Human body imaging
- Understanding the universe
- Increasing global agricultural yield
- Sequencing microbiome genome data


8) (3 pts) What are the values of list1, list2, and list3 after the code below executes? **Write your answers in the corresponding boxes below.**





```
script variables list1 list2 list3
set list1 to list 1 2
set list2 to list1
set list3 to map 1 + over list2
add 3 to list2
```


list1
[1, 2, 3]



list2
[1, 2, 3]



list3
[2, 3]

9) (4 pts) Inspired by our Tic-Tac-Toe lab, we wrote a block called . It should return True if the given player (input #1; X or O) can win on the next turn if they were to make the next move, using the given Tic-Tac-Toe board (input #2). But we're not quite sure if our block is working properly. We need your help with testing the block.

Complete our test cases by filling in the blanks in the  blocks below. Each test case should report the corresponding output (for the inputs for the first test case when inputted into  should report  and the second test case should report ). **You must write directly in the empty squares below.** When you're finished, each square in each board should either contain an X, contain an O, or remain empty.

test  **w/inputs**

list  

expecting output  

NOTE #1: Each section of three blanks in the “new 3x3 board” block represents a row on our Tic-Tac-Toe board.

NOTE #2: A player wins by drawing three X's or three O's in a row, either horizontally, vertically, or diagonally.

There were many possible correct answers to question #9, too many to list here.

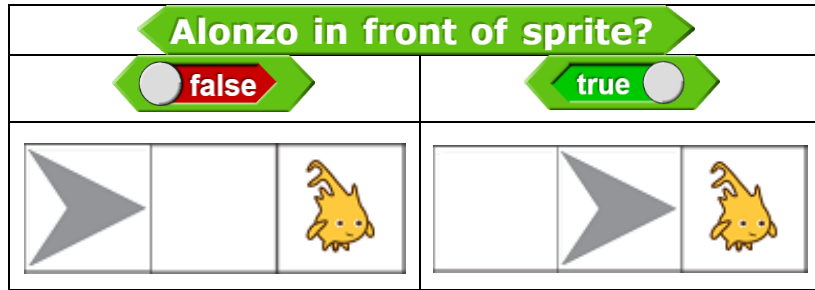
Question 10: Alonzo's Playground (10 points total, 5 each part, 20 min.)

We want to give our sprite a grid for our arrow sprite to travel and play around in. Our sprite can move into empty white squares, but cannot move into black squares or off the grid. The directionality depends on the orientation of the sprite. See the picture below for examples.



ID Card Number: _____

Also, we don't want to hit Alonzo! If Alonzo is directly in front of our sprite, we want our sprite to *stop moving completely*.

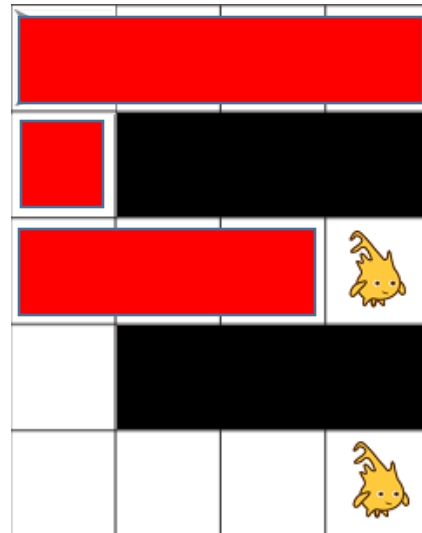


For each of the following scripts, shade in all the squares that the arrow sprite visits. Assume that the arrow sprite starts in the top-left corner, facing right, at the beginning of each script.

a)

```

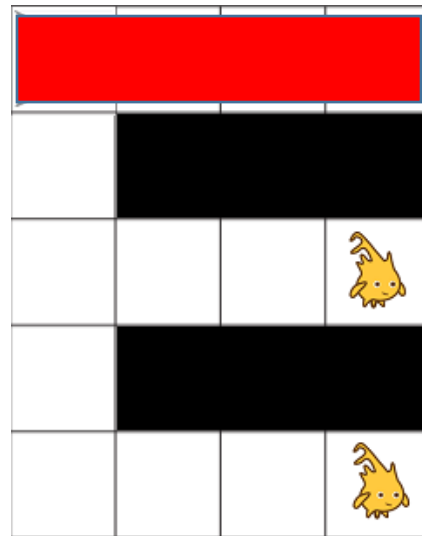
when clicked
  forever
    if square to the left of sprite empty?
      rotate left
    if square in front of sprite empty?
      move forward one square
    else
      rotate left
    if Alonzo in front of sprite?
      stop all
  
```



b)

```

when clicked
  forever
    if square to the left of sprite empty?
      rotate left
      move forward one square
    else
      if square in front of sprite empty?
        move forward one square
      else
        if Alonzo in front of sprite?
          stop all
  
```



Question 11: Do You Believe in Magic? (15 points total, 30 min.)

Your best friend comes back from a trip to Alonzoville and surprises you with a souvenir, the code below! Unfortunately, your friend forgets to tell you what it does. So let's investigate together....

```

+ Word + Magic + input +
script variables first rest beginning end
if length of input = 1 or input = 
  report input
set first to letter 1 of input
set rest to
  Word Magic list → word all but first of word → list input
set beginning to 
set end to rest
for each item of word → list rest
  if first > item
    set beginning to join beginning letter 1 of end
    set end to list → word all but first of word → list end
report join beginning first end
  
```

NOTE: When Snap! compares letters, it treats letters that come earlier in the alphabet as less than letters that come later in the alphabet. For some examples, see the calls below.

A < B → true

B < A → false

Y > X → true

X > X → false

a) (4 pts) What does **Word Magic** **EPA** report? Write your answer in the box provided to the right.

AEP

b) (4 pts) If we want **Word Magic** **computer** to report the word *utrpomec*, what will we have to change? (Hint: Don't trace through the code with this very long input. Instead, think about what Word Magic does to all inputs, based on your answer to part (a).)

- Swap the order of **set beginning to join beginning letter 1 of end** and **set end to list → word all but first of word → list end**
- Change **first > item** to **item > first**
- Change **join beginning first end** to **join end first beginning**
- All of the above will cause Word Magic to report *utrpomec*
- None of the above will cause Word Magic to report *utrpomec*
- No change is necessary; Word Magic already reports *utrpomec*

ID Card Number: _____

c) (3 pts) What is the worst case runtime of Word Magic, as a function of the length of the input word?

<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

d) (4 pts) We decide to use Word Magic to perform a magic trick. Assume that we create a global variable “Magical,” click the green flag, and wait 60 seconds.

The code consists of three main scripts:

- When clicked:** A script starting with a green flag icon, followed by a 'set Magical to Word Magic X' block, and a 'broadcast Magic Show' block.
- When I receive Magic Show (Left):** A script starting with 'when I receive Magic Show', followed by 'wait pick random 1 to 5 secs', a 'repeat until' loop containing 'Magical = Word Magic B or Magical = Word Magic C', a 'broadcast Magic Show' block, and finally 'set Magical to Word Magic A'.
- When I receive Magic Show (Right):** A script starting with 'when I receive Magic Show', followed by 'wait pick random 1 to 5 secs', a 'repeat until' loop containing 'Magical = Word Magic A or Magical = Word Magic C', a 'broadcast Magic Show' block, and finally 'set Magical to Word Magic B'.
- When I receive Magic Show (Bottom):** A script starting with 'when I receive Magic Show', followed by 'wait pick random 1 to 5 secs', a 'repeat until' loop containing 'Magical = Word Magic A or Magical = Word Magic B', a 'broadcast Magic Show' block, and finally 'set Magical to Word Magic C'.

What are all of the possible values of Magical after 60 seconds have elapsed? Write your answer in the box provided below. There may be more space than you need.



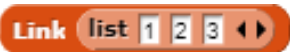

X

Question 12: Lists Inside of Lists Inside of Lists Inside of... (10 pts, 20 min.)


A *linked list* is defined as a list with exactly two items: the first item is a value (i.e., a number or text) and the second item is another linked list. A linked list may also be empty, in which case it contains exactly zero items.

Write a function, “Link,” which converts normal Snap! lists into linked lists. It takes a list as input, and reports the corresponding linked list as output.

See the table below for a few sample calls to Link.

Function Call	Output
	
	

For full credit, your solution may **only** use recursion (no loops or HOFs) and must **not** modify the input list. Your solution must fit on the lines below...**but you may not need to use all of the lines.**



```

+ Link + input list +
if empty? input list
  report input list
report
  item 1 of input list in front of
  list Link all but first of input list

```

(Hint: Remember what the “in front of” block does. See the sample call to the right.)



1 in front of list 2 3



1 1
2 2
3 3
length: 3

ID Card Number: _____

Question 13: If You Thought Expand Was Fun... (15 pts total, 20 min.)

In lab you saw the “Expand” block, which modified input sentences by repeating certain words. Now, let’s write a slightly different version of Expand: Expand All.

Expand All takes as input an alternating sequence of numbers and words, beginning with a number and ending with a word. It returns a sequence of only words, in which each word is repeated x times, where x is the number directly preceding the word in the input sequence. For example, if we enter “2 Hello 1 Goodbye” as the input, then $x = 2$ for “Hello” and $x = 1$ for “Goodbye.” Expand All will report “Hello Hello Goodbye” as output.



You have been given two new blocks to help you write Expand All. “Generate Even Numbers Between (X) And (Y)” takes as input two numbers (X and Y), and outputs a list of all even numbers between X and Y , inclusive. “Repeat Word” takes as input a word and a number n , and reports the word repeated n times.



Fill in the code on the next page to implement Expand All using **only Higher Order Functions (HOFs)**. Write all answers in the boxes provided. Anything written outside the boxes will not be scored.

The code in the Scratch script area is as follows:

```
+ expand + all + sentence +
script variables input list
set input list to sentence to list sentence
report map repeat word over
generate even numbers between 1 and
```

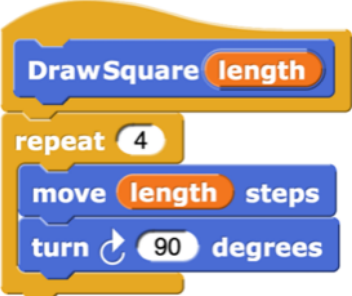

Callout boxes with arrows pointing to the code:

- Top box: join words [] []
- Right box: item [] - 1 of input list
- Bottom-left box: item [] of input list
- Bottom-right box: length of input list

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:

<p>Draw-Square (LENGTH)</p> <pre>repeat (4) move (LENGTH) steps turn-right (90) degrees</pre> 	<p>(N)!</p> <pre>if N = 0 report (1) report (N * (N - 1)!)</pre> 
--	--

- 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:

things

```
1 life
2 liberty
3 pursuit of happiness
length: 3
```




- **(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)).**

nested

```
1 Love
1 2 5
length: 2
2 Hate
2 2 4
length: 2
3 The
3 2 10
length: 2
length: 3
```



- 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.

