UC Berkeley's CS10 Spring 2019 Quest ANSWERS – Prof. Dan Garcia

Your Name (first last) *SID* *Lab TA's Name*

← *Name of person on left (or aisle)* *Name of person on right (or aisle)* →

What's that Smell? Oh, it's Potpourri! (2 pts for 1-6, we drop lowest one)

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

Question 1: Which of the following is a **true** statement regarding Abstraction? (select ONE)

○ A freight company uses *generalization*, since they don't care what's inside the freight container, as long as it adheres to the appropriate specifications (e.g., within weight limits, won't leak, and has no combustibles).

FALSE! This is certainly abstraction, but it's *detail removal*, not *generalization.* Said another way, the freight company doesn't need to know what's below the *abstraction barrier*, or *below the abstraction line*.

○ Harry Beck took the map of the London Underground that had stations equally spaced out and lines only at at 0º, 45º, 90º and 135º and made it more geographically accurate, since *details matter*.

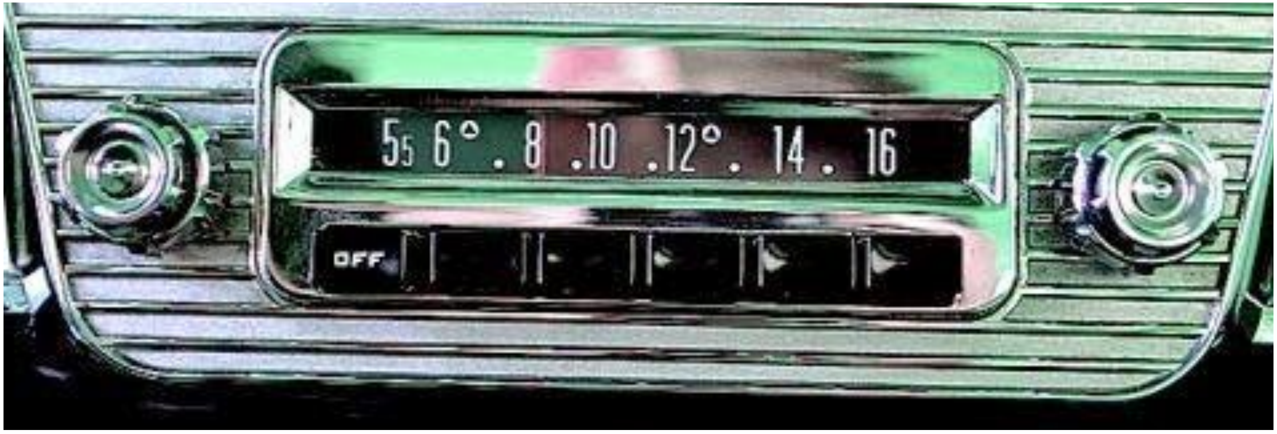
FALSE! Just the opposite, he took the geographically accurate map and made an abstract view of it, *detail removal*.

○ A farmer who writes "put the <animal> food in the <animal dish>" is using *detail removal*.

FALSE! This is *generalization*, not *detail removal*.

○ A person from 1920 could still use the car radio in today's car thanks to Abstraction.

FALSE! They might be able to *drive* because the interface for driving has mostly stayed the same (wheel and accelerator pedal on the right and brake pedal on the left of that) but the interface for radios has significantly changed from the radios of days gone by, so there's no way that someone used to that interface could use the radio in today's car. (actually, radios didn't make it into cars until the 1950s)

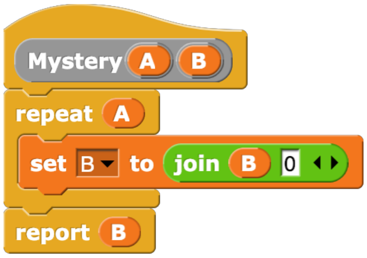


● None of the Above.

Question 2: What is $18_{16} \div 8_{10}$? (select ONE)

○	○	●	○	○	○
1_2	10_2	11_2	100_2	101_2	110_2

$18_{16} = 1*16^1 + 8*16^0 = 1*16 + 8 = 24$ divided by $8 = 3_{10} = 1*2^1 + 1*2^0 = 11_2$



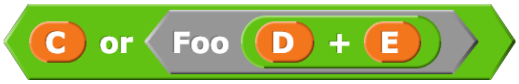
```
Mystery A B
repeat A
    set B to join B 0
report B
```

Question 3: What does Mystery 3 5 report? (select ONE)

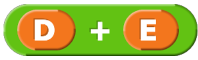
○	○	○	○	○	○	●	○	○	○	○
0	30	50	505050	3030303030	300000	5000	5550	333330	Error	Infinite Loop

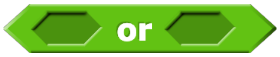
The loop repeats A→3 times, so B gets 0 added to the right 3 times. It starts with 5, so it's 5000.

Question 4: What is the *Domain* and *Range* of **Foo**?
The expression does not cause an error. (select ALL that apply)

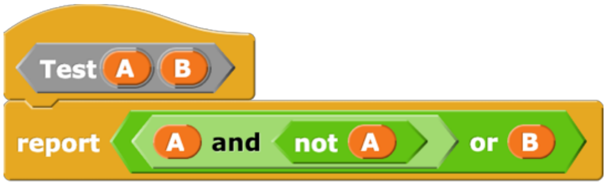


The Domain of **Foo** is…					The Range of **Foo** is…				
■	□	□	□	□	□	□	□	■	□
numbers	words	sentences	Booleans	lists	numbers	words	sentences	Booleans	lists

Foo takes the result of , and we know that has to be a number, so the Domain is *numbers*.

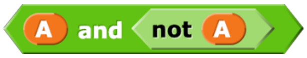
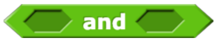
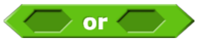
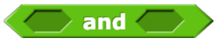
Foo's the result is passed into  whose domain is Booleans, so the range is *Booleans*.

(The block here is used for Questions 5 & 6)

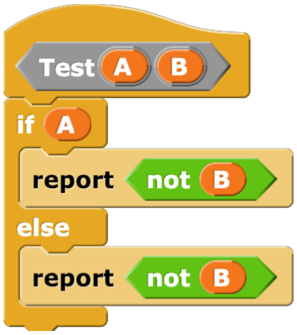
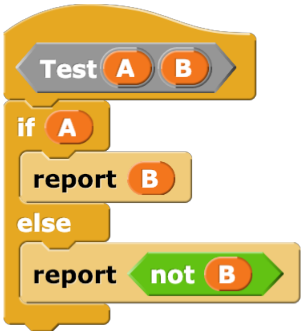
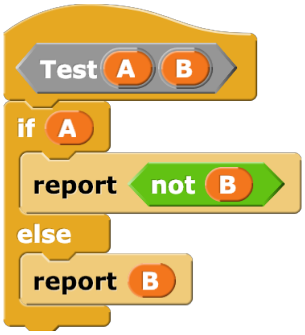
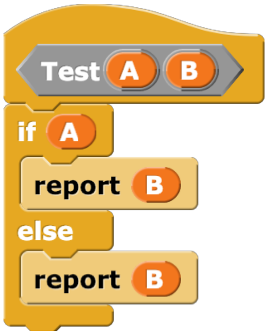


Question 5: If **A** and **B** are Booleans, and the output from **Test** is true, which can you say *for sure*? (select ALL that apply)

□	■	□	□	□
A must be true	**B** must be true	**A** must be false	**B** must be false	None of these

 is always false, since both inputs of  have to be true for the expression to be true – regardless of the value of **A**, one of those inputs will be true and one of those inputs will be false. Another thing we realize is that  is always "looking" for a true – if it finds it, it reports true, otherwise, it'll be false. Since the  is false, it needs to look at **B**, which means effectively if **B** is true, then **Test** will be true, and if **B** is false, then **Test** will be false. So really, **Test** will return whatever **B** is, and therefore we know if the output of **Test** will be true, then **B** is true.

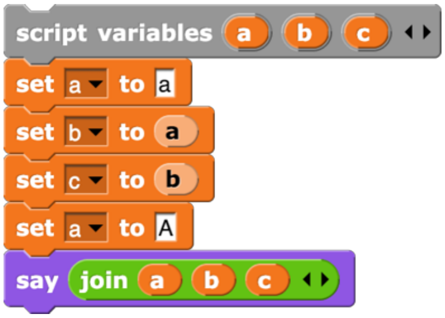
Question 6: Which of the following is equivalent to the original **Test** block? (select ONE)

○	○	○	●	○
				None of these

Using the logic from the earlier problem, **Test** will return whatever **B** is, so it has to return **B** either way.

Question 7: *Say say say…say it isn't so…* (2 pts)

This script is intended to say some letters. What gets said? (select ONE)



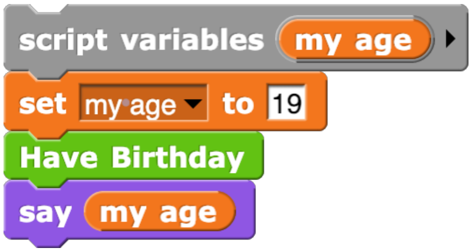
```
script variables a b c
set a to a
set b to (a)
set c to (b)
set a to A
say (join (a) (b) (c))
```

○	○	○	○	●	○	○
aaa	abc	AAA	Abc	Aaa	No Error, but none of these	Error (since you can't set variable a twice)

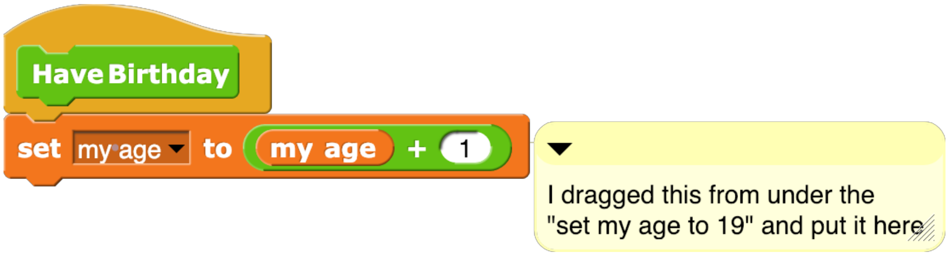
After this block…	a will be…	b will be…	c will be…
script variables a b c	Undefined	Undefined	Undefined
set a to a	a	Undefined	Undefined
set b to (a)	a	a	Undefined
set c to (b)	a	a	a
set a to A	A	a	a

Question 8: *Happy birthday to me!!…* (3 pts)

What gets said if I run the script on the left? *Note: We have not yet defined any global variables.* (select ONE)



```
script variables my age
set my age to 19
Have Birthday
say (my age)
```



```
Have Birthday
set my age to ((my age) + 1)
```

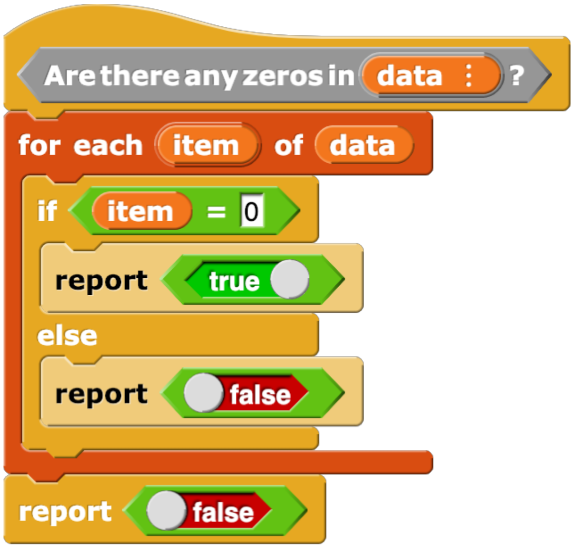
I dragged this from under the "set my age to 19" and put it here

○	○	●
19	20	Error (a variable of name 'my age' does not exist in this context)

The variable `my age` is not in scope in `Have Birthday` (it's only in scope for the script on the left), so it's an error.

Question 9: *Beethoven was a good composer…* (5 pts, one each)

You write a block to determine if there are any zeros in a list of numbers (with at least one element), if so it should return true. Will it return the correct value for different inputs?



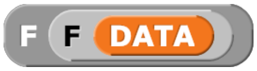
```
Are there any zeros in (data ⋮) ?
for each (item) of (data)
    if <(item) = 0>
        report <true>
    else
        report <false>
report <false>
```

There is exactly one element in the list, and it is zero.	yes ● ○ no
There is exactly one element in the list, and it is not zero.	yes ● ○ no
The list has all zeros.	yes ● ○ no
The list has no zeros.	yes ● ○ no
The code works correctly for all inputs.	yes ○ ● no

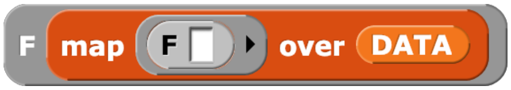
The code only gets to look at the first item of `data`, since the if statement will report one way or another and the procedure will end, without looking at the whole list of numbers. Effectively, a better name of this procedure is: “Is the first item of `data` a zero?”. So, if you’re only looking at the first element, the first four are true! But the code certainly doesn’t work correctly for all inputs…

Question 10: *Two maps are better than one! (or are they?)* (5 pts)

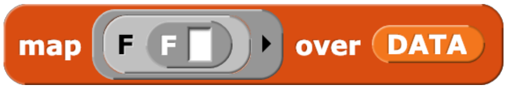
Of the six expressions:

a) 

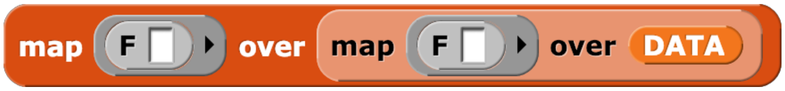
`F F DATA`

b) 

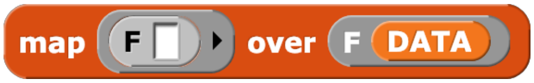
`F map F ▢ over DATA`

c) 

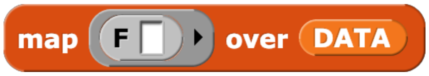
`map F F ▢ over DATA`

d) 

`map F ▢ over map F ▢ over DATA`

e) 

`map F ▢ over F DATA`

f) 

`map F ▢ over DATA`

	a	b	c	d	e
b	□				
c	□	□			
d	□	□	🟥		
e	□	□	□	□	
f	□	□	□	□	□

…which are *always equivalent* for **all** DATA lists and **all** mappers F ▢? Said another way, which pairs will *always* report the same value? (select ALL that apply; selecting a particular box means you are declaring that the expression in the row will always have the same value as the expression in the column **for all input.**

This question was meant to test understanding of domain and range. **`map`** is a higher-order function that takes a function **`F`** and a list of **`data`** and returns a new list in which every element of **`data`** has the function **`F`** applied to it. If we were to visualize a list as set of parentheses, so the list of three numbers `1`, `2`, `3` would be represented as `( 1 2 3 )`, then here's what the value of each expression would be if **`data`** were the list `( 1 2 3 )` and the function **`F`** were shown at the top.

	list ▢ ◂▸	is ▢ a Boolean ?
a	`( ( ( 1 2 3 ) ) )`	`true`
b	`( ( (1) (2) (3) ) )`	`false`
c	`( ((1)) ((2)) ((3)) )`	`( true true true )`
d	`( ((1)) ((2)) ((3)) )`	`( true true true )`
e	`( ( ( 1 2 3 ) ) )`	Error expecting list but getting Boolean
f	`( (1) (2) (3) )`	`( false false false )`
Same?	**c=d** and **a=e**	**c=d**

This isn't a proof that c and d are the same (but is an existence proof that none of the others are), so let's take a closer look at it.

Expression **c** returns a list in which every element **`x`** of data is **`F(F(x))`**. So in our simple list case, if **`data`** were the list `( 1 2 3 )`, c would evaluate to `( F(F(1)) F(F(2)) F(F(3)) )`.

Expression **d** has a nested map. After the first map, every element **`x`** of data is **`F(x)`**. So in our simple list case, if **`data`** were the list `( 1 2 3 )`, the first map would evaluate to `( F(1) F(2) F(3) )`.That's passed into the second map, in which every element **`x`** of data becomes **`F(F(x))`**. So in our simple list case, the second map would evaluate to `( F(F(1)) F(F(2)) F(F(3)) )`.
Therefore **c** and **d** will always be the same (and no other two will always be).