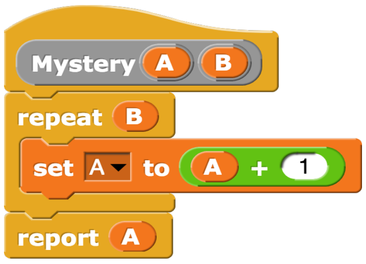
CS10 Fall 2018 Quest Answers

Question 1: Which of the following is a *true statement* regarding Abstraction? **An *abstraction barrier* allows us to use something without needing to know how it is built.**

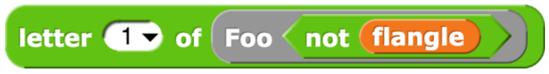
Question 2: What is $11_{16} - 11_{2}$? $11_{16} = 1*16 + 1*1 = 17_{10}$, 11_{2}, $=1*2+1*1=3_{10}$, so $17_{10}-3_{10}$ = **14_{10}.**

Question 3: What does **Mystery** report, if B is a non-negative integer (i.e., 0, 1, 2, …)?
1 gets added to A exactly B times, so A + 1+1+…+1 (B times) = A+1*B = **A+B**

Question 4: What is your guess as to the *Domain* and *Range* of **Foo**?
The expression doesn't cause an error. `not` returns a Boolean so the domain of Foo is **Booleans**. `letter (1) of ( )` takes a sentence (which is a superset of words which is a superset of numbers) so the range is **numbers, words and sentences**.

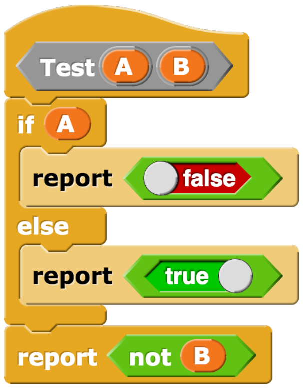


```
Mystery (A) (B)
repeat (B)
    set [A] to ((A) + (1))
report (A)
```



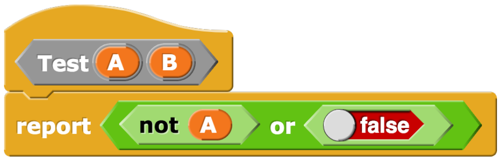
```
letter (1) of (Foo <not <flangle>>)
```

Question 5: If the output from **Test** is true, which can you say *for sure*? The second report (i.e., the value of B) is ignored since the initial if A either returns false or true. So if it returned true, **A must be false**.

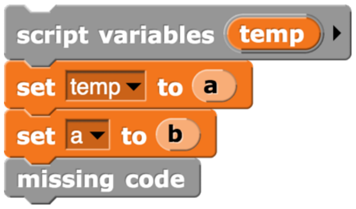


```
Test (A) (B)
if (A)
    report <false>
else
    report <true>
report <not (B)>
```

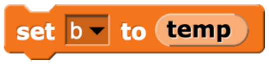
Question 6: Which of the following are the same as the original **Test** block? The block is effectively "not A" since **Test** returns true when A is false and false when A is true. Only not(A) or false is the same as not(A).



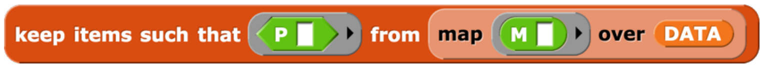
```
Test (A) (B)
report <<not (A)> or <false>>
```



```
script variables (temp)
set [temp] to (a)
set [a] to (b)
missing code
```

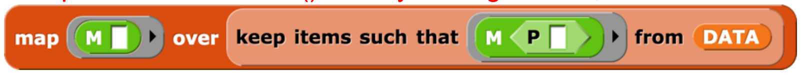
Question 7: This script is intended to exchange the values of the variables **a** and **b** using the temporary variable **temp**. Which of the following can be used to replace `missing code` so the script works as intended? (select ONE)  `set [b] to (temp)`

Question 8: If we were given three functions:
F(x) = x^2
G(x) = x – 7
H(x) = x + 5
…and you wanted to calculate:
$(x – 7)^2 + 5$
…how would you compose the three functions to get that? **H(F(G(x)))**, since "x-7" happens first, then squared, then x+5.



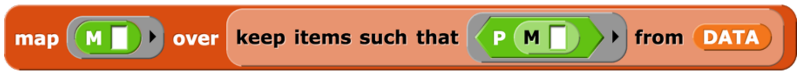
```
keep items such that (P [ ]) from (map (M [ ]) over (DATA))
```

Question 9: We want to compute the following cascade of **map** with mapping function **M()** and **keep** with predicate **P()**, but someone "glues" the **map** and **keep** together in the wrong order! Let's try to change the inputs to **map** and **keep** to make it work. Which works, which can potentially cause a domain/range error, and which doesn't cause an error but is probably a wrong answer? Imagine if DATA were "a list of lists of numbers (i.e., a 2D table of numbers)". M(x) is "item(1) of x" and P(x) is "x < 5". So the original code was meant to grab the first column of a table (i.e., first number from each inner list) and keep all the numbers from that column less than 5. It doesn't make sense to ask if a list is less than 5. Note that three of the options below have P() directly looking at a list, and in this example list < 5 is a Domain and Range (i.e., D&R) error.



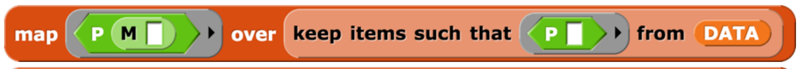
```
map (M [ ]) over (keep items such that (M (P [ ])) from (DATA))
```

D&R error because P is being run on DATA directly



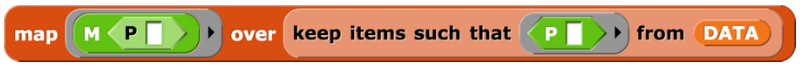
```
map (M [ ]) over (keep items such that (P (M [ ])) from (DATA))
```

This works!



```
map (P (M [ ])) over (keep items such that (P [ ]) from (DATA))
```

D&R error because P is being run on DATA directly



```
map (M (P [ ])) over (keep items such that (P [ ]) from (DATA))
```

D&R error because P is being run on DATA directly