CPSC 100

Computational Thinking

Intro to Programming

Instructor: Firas Moosvi

Department of Computer Science

University of British Columbia

Agenda

- Revisit Comparing Loops
- Code Explanations
- Code Tracing
- Debugging
- Modulo Operator

Learning Goals

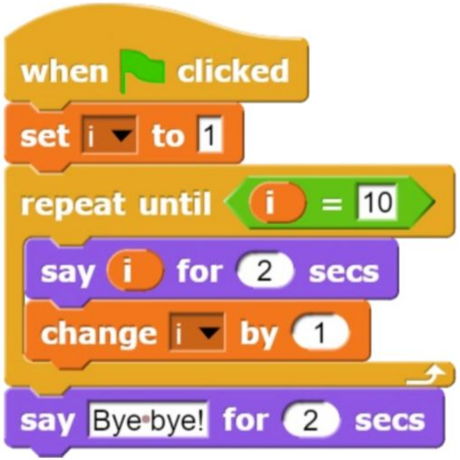
After this today's lecture, you should be able to:

- Identify and understand the differences between **different types of loops** (e.g. repeat, repeat until, for)
- **Trace** through code using sequences of instructions, variables, loops, and conditional statements in short programs
- **Describe in English** what a block of Snap! code does.
- **Evaluate** if a given snap code block correctly implements an algorithm.

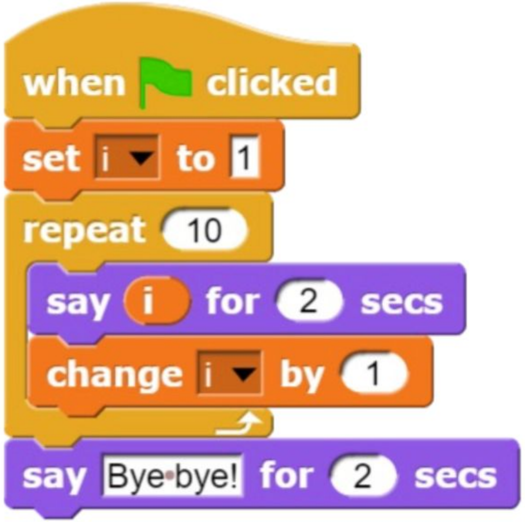
Course Admin

Comparing Loops

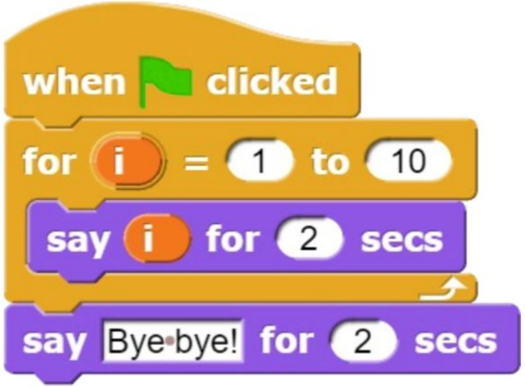
What's the difference between these loops?



```
when ⚑ clicked
set i to 1
repeat until (i) = 10
    say (i) for 2 secs
    change i by 1
say Bye-bye! for 2 secs
```

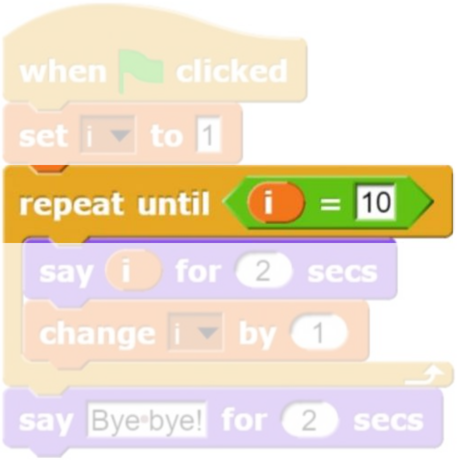


```
when ⚑ clicked
set i to 1
repeat 10
    say (i) for 2 secs
    change i by 1
say Bye-bye! for 2 secs
```

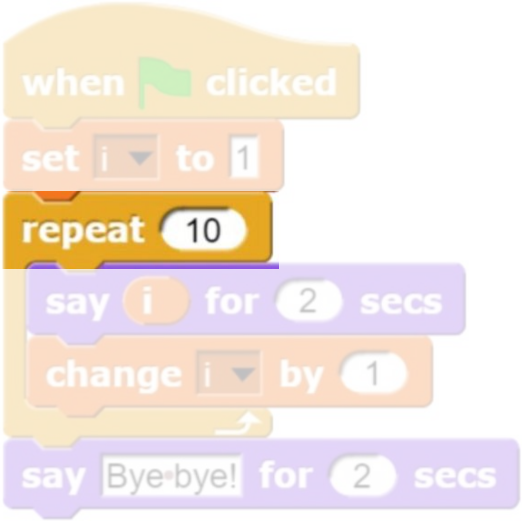


```
when ⚑ clicked
for (i) = 1 to 10
    say (i) for 2 secs
say Bye-bye! for 2 secs
```

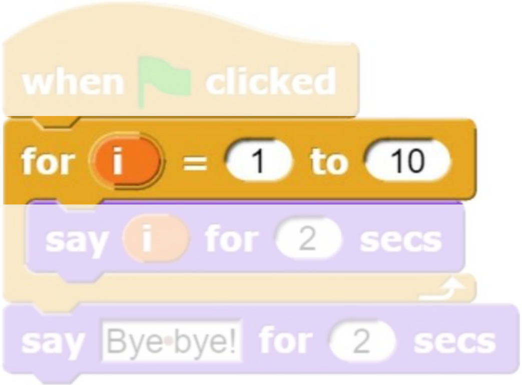
What's the difference between these loops?



```
when 🏳 clicked
set i ▾ to 1
repeat until < (i) = 10 >
    say (i) for (2) secs
    change i ▾ by (1)
say Bye bye! for (2) secs
```



```
when 🏳 clicked
set i ▾ to 1
repeat (10)
    say (i) for (2) secs
    change i ▾ by (1)
say Bye bye! for (2) secs
```



```
when 🏳 clicked
for (i) = (1) to (10)
    say (i) for (2) secs
say Bye bye! for (2) secs
```

Code Explanation

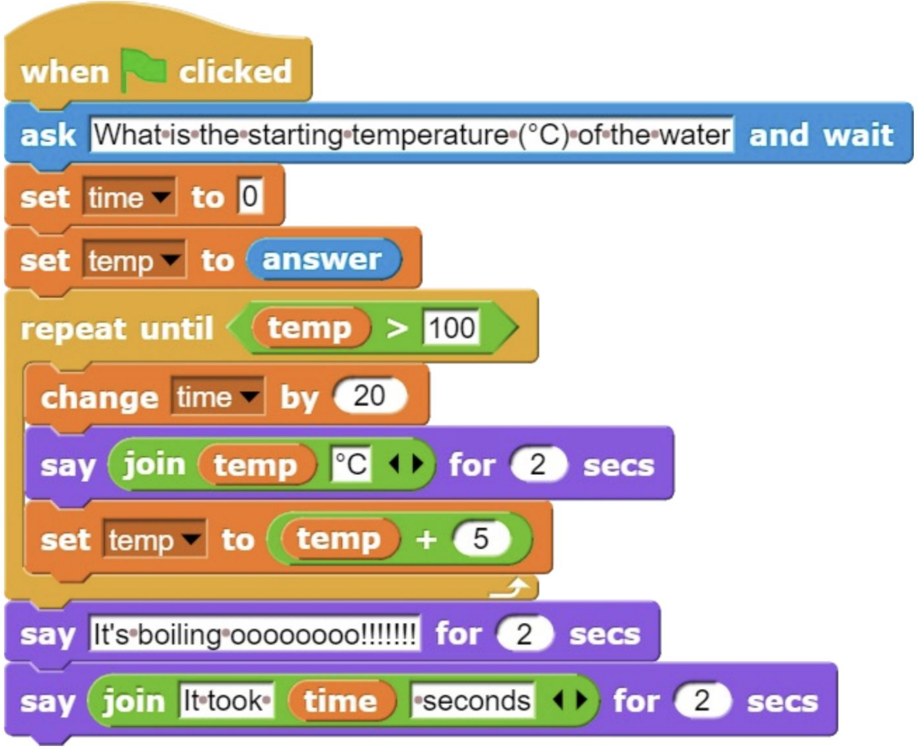
What does this code block do?

Describe the code in terms of input and output and what is being done.

The block consumes....

It does...

It produces/displays/reports…



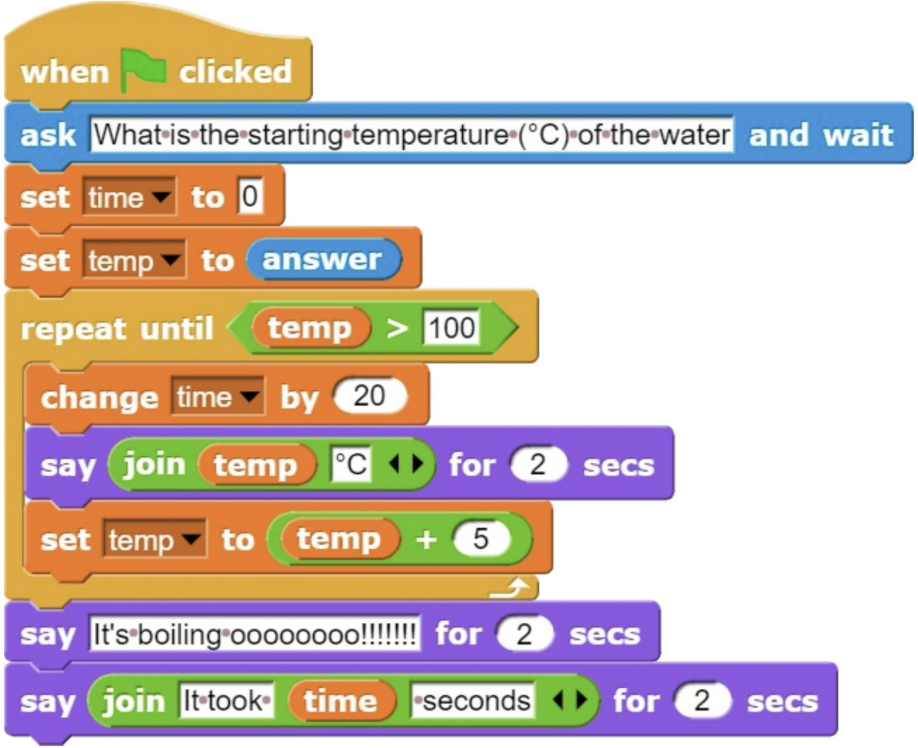
```
when 🏁 clicked
ask [What is the starting temperature (°C) of the water] and wait
set [time] to [0]
set [temp] to (answer)
repeat until <(temp) > [100]>
    change [time] by (20)
    say (join (temp) [°C] ◂▸) for (2) secs
    set [temp] to ((temp) + (5))
say [It's boiling oooooooo!!!!!!!] for (2) secs
say (join [It took ] (time) [ seconds] ◂▸) for (2) secs
```

What is the purpose of this block?

The block consumes the starting temperature for water

It increases the temperature and time

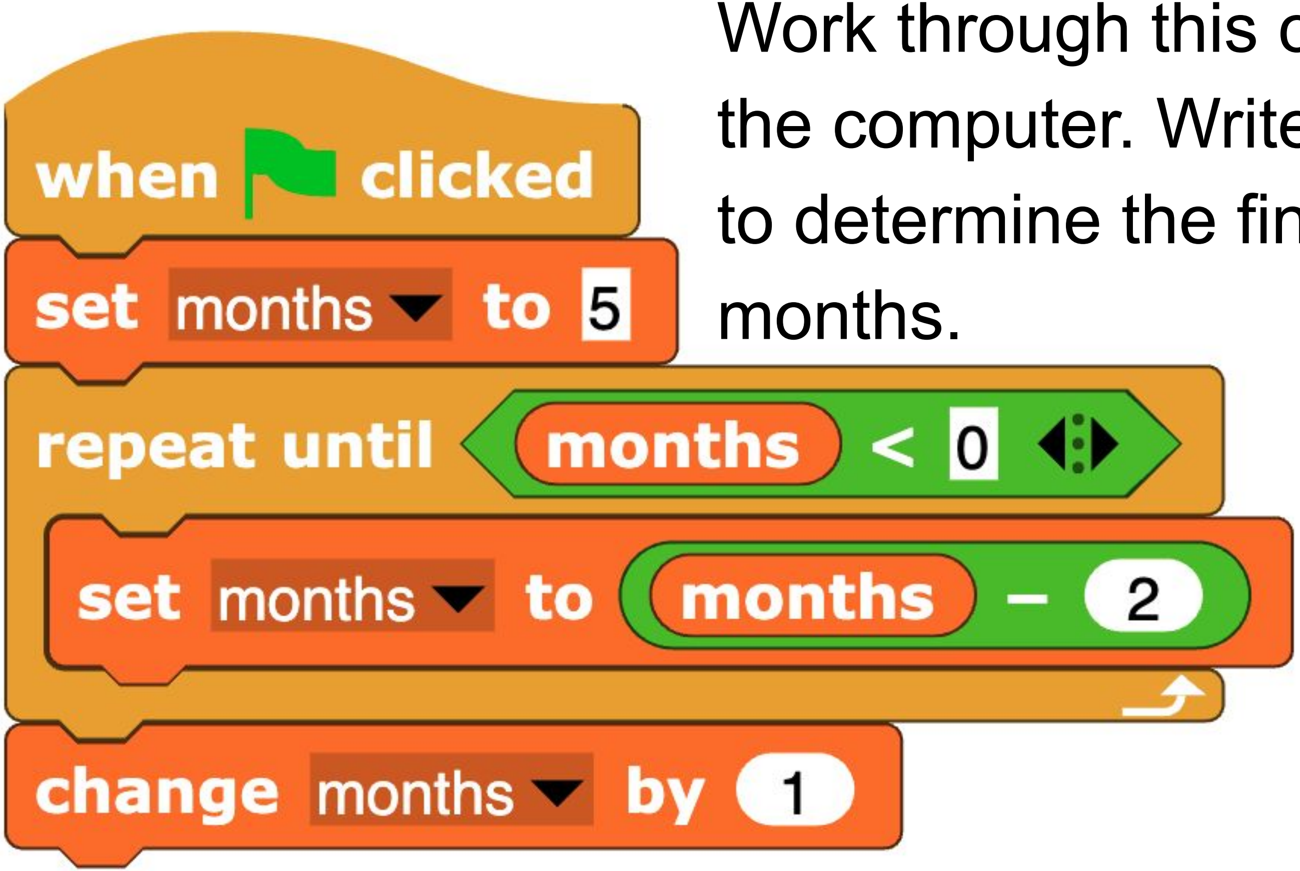
It produces the time it takes to get the water from the starting temperature to 100 degree Celsius



```
when ⚑ clicked
ask What is the starting temperature (°C) of the water and wait
set time to 0
set temp to answer
repeat until temp > 100
    change time by 20
    say join temp °C for 2 secs
    set temp to temp + 5
say It's boiling oooooooo!!!!!!! for 2 secs
say join It took time seconds for 2 secs
```

Code Tracing Demo

Activity - Step through this code



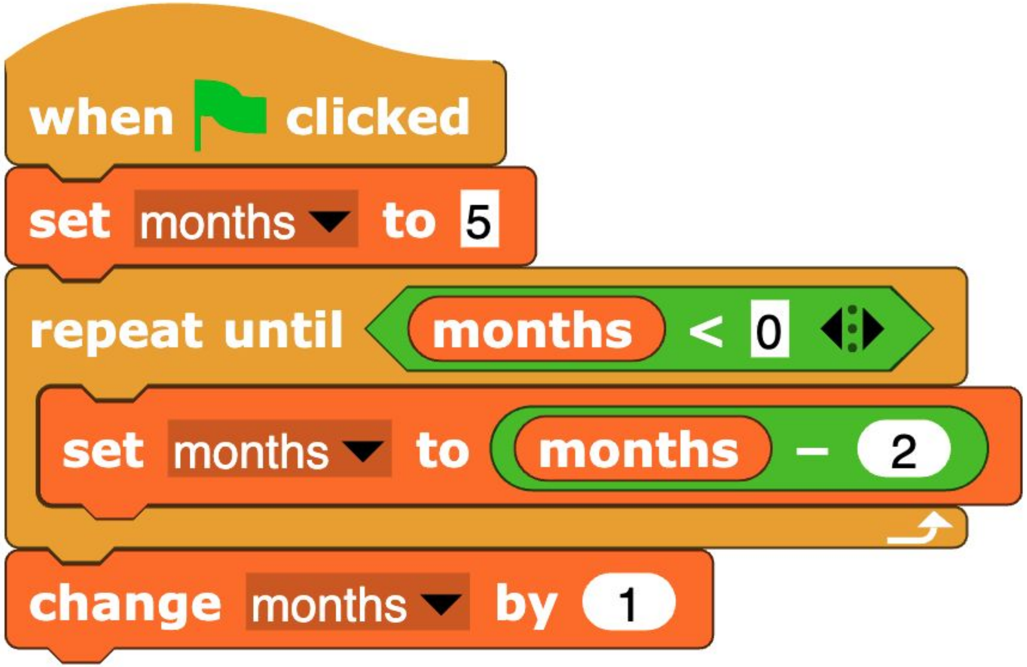
```
when 🏁 clicked
set months ▾ to 5
repeat until < (months) < 0 ◂▸ >
    set months ▾ to ((months) − 2)
change months ▾ by 1
```

Work through this code as if you're the computer. Write down each step to determine the final value of months.

Activity - Step through this code

Step 1: Write down the variables you want to track.

months □

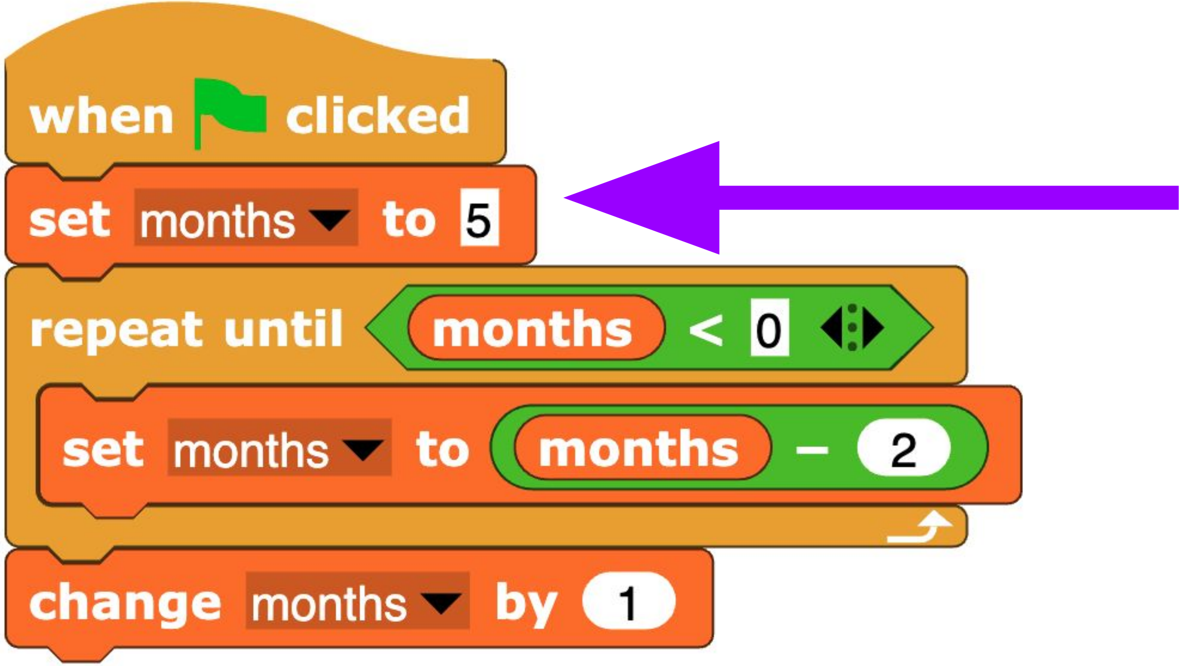


```
when ⚑ clicked
set months to 5
repeat until months < 0
    set months to months − 2
change months by 1
```

Activity - Step through this code

Step 2.0: Write down the value of each variable before entering the loop/repeat block.

months $\boxed{5}$

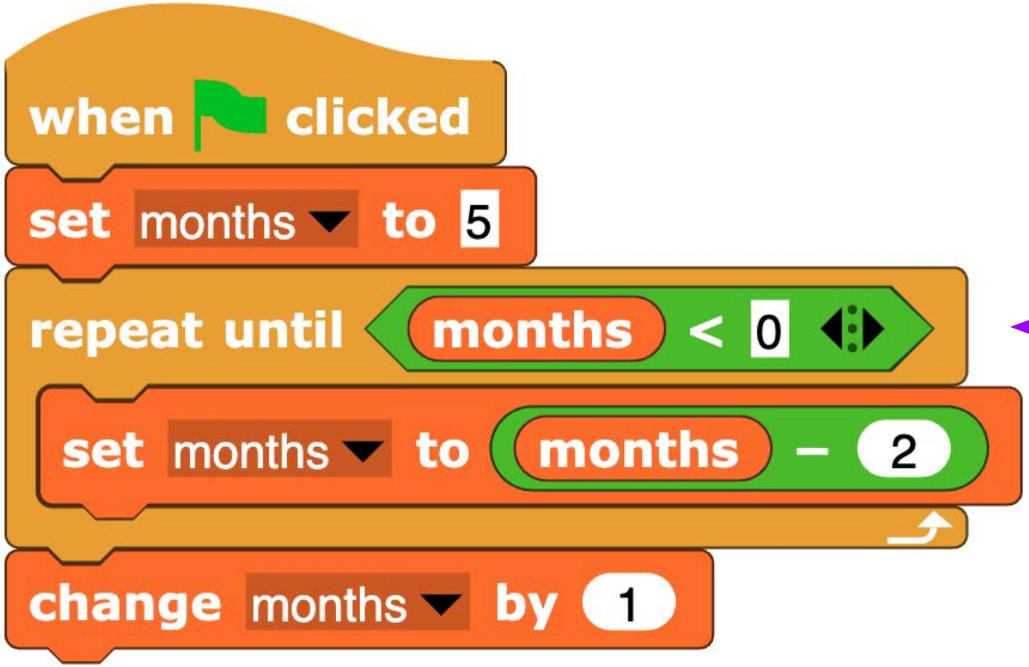


```
when 🏳 clicked
set months ▾ to 5
repeat until ( months < 0 ◂▸ )
    set months ▾ to ( months − 2 )
change months ▾ by 1
```

Activity - Step through this code

Step 2.1: Go through each step of the loop, keep track of the variable and its value.

months 5



```
when ⚑ clicked
set months to 5
repeat until <(months) < 0>
    set months to ((months) - 2)
change months by 1
```

Is the condition true?

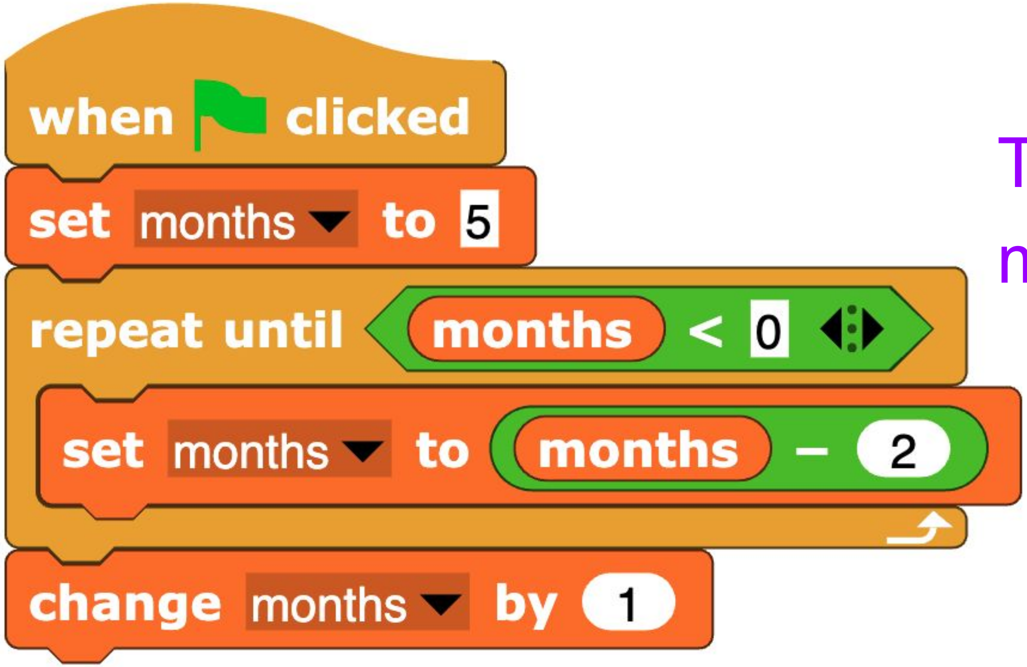
No → Execute code inside the loop.

Yes → Exit the loop, execute the first line after the loop.

Activity - Step through this code

Step 2.2: Go through each step of the loop, keep track of the variable and its value.

months 5



```
when ⚑ clicked
set months ▾ to 5
repeat until < (months) < 0 ◂▸ >
    set months ▾ to ((months) − 2)
change months ▾ by 1
```

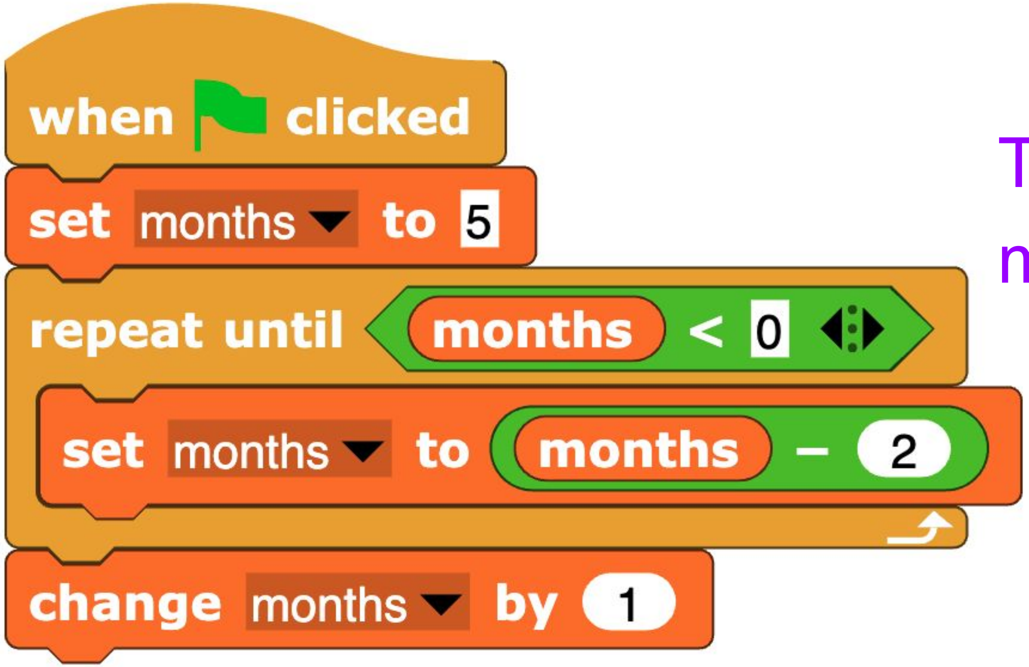
The value stored inside the months variable changes:

5 - 2 = 3

Activity - Step through this code

Step 2.2: Go through each step of the loop, keep track of the variable and its value.

months $\boxed{3}$



```
when green flag clicked
set months to 5
repeat until months < 0
    set months to months - 2
change months by 1
```

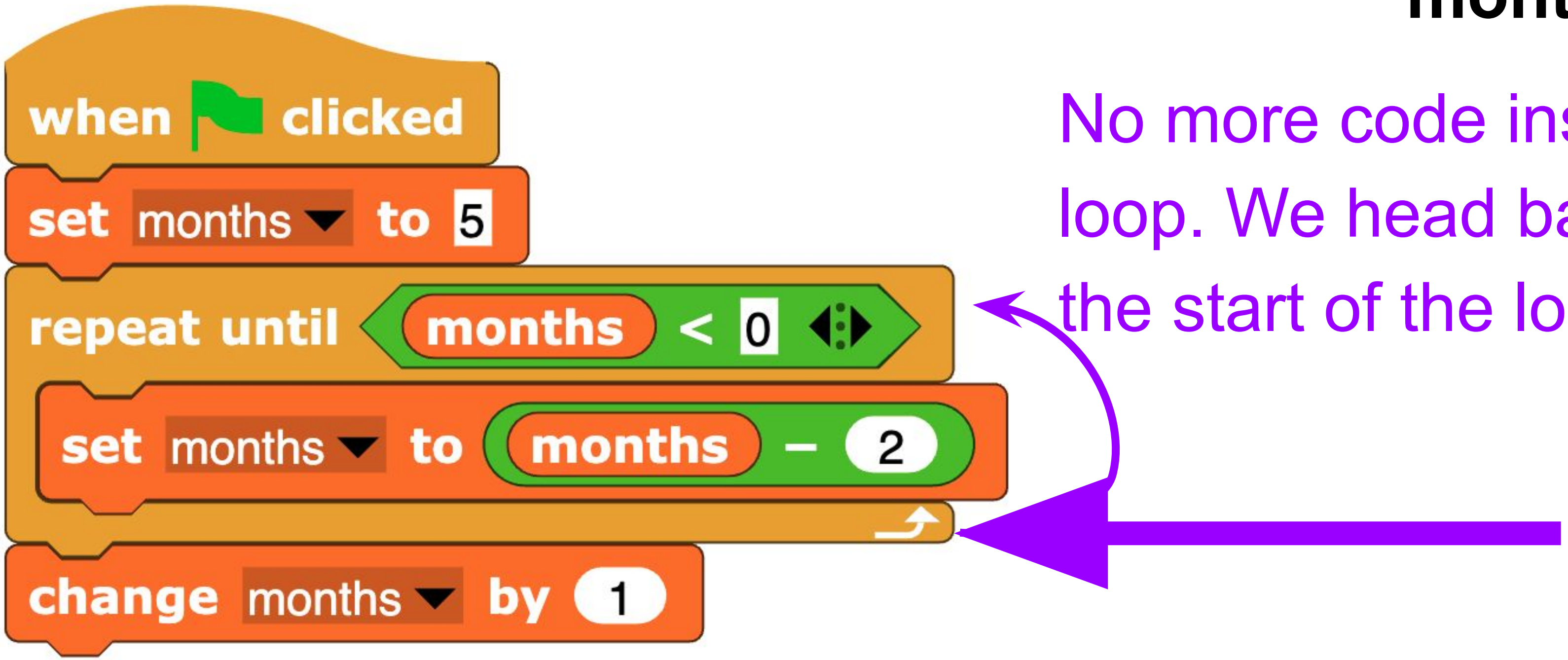
The value stored inside the months variable changes:

5 - 2 = 3

Activity - Step through this code

Step 2.3: Go through each step of the loop, keep track of the variable and its value.

months $\boxed{3}$



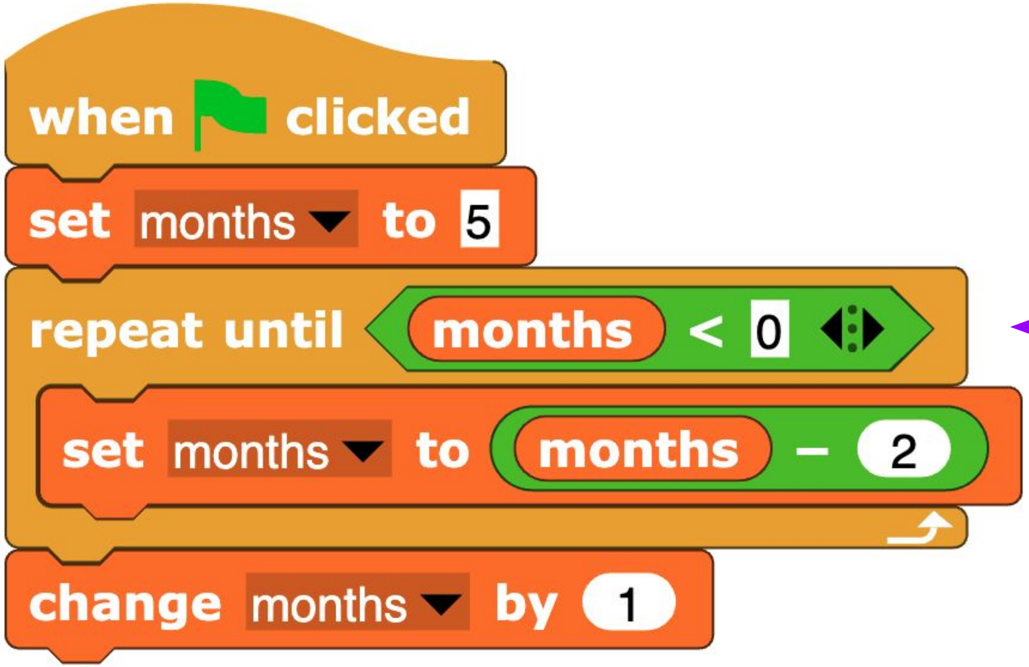
```
when ⚑ clicked
set months ▾ to 5
repeat until months < 0
    set months ▾ to months − 2
change months ▾ by 1
```

No more code inside the loop. We head back up to the start of the loop.

Activity - Step through this code

Step 2.4: Go through each step of the loop, keep track of the variable and its value.

months $\boxed{3}$



```
when ⚑ clicked
set months ▾ to 5
repeat until ( months < 0 )
    set months ▾ to ( months − 2 )
change months ▾ by 1
```

Is the condition true?

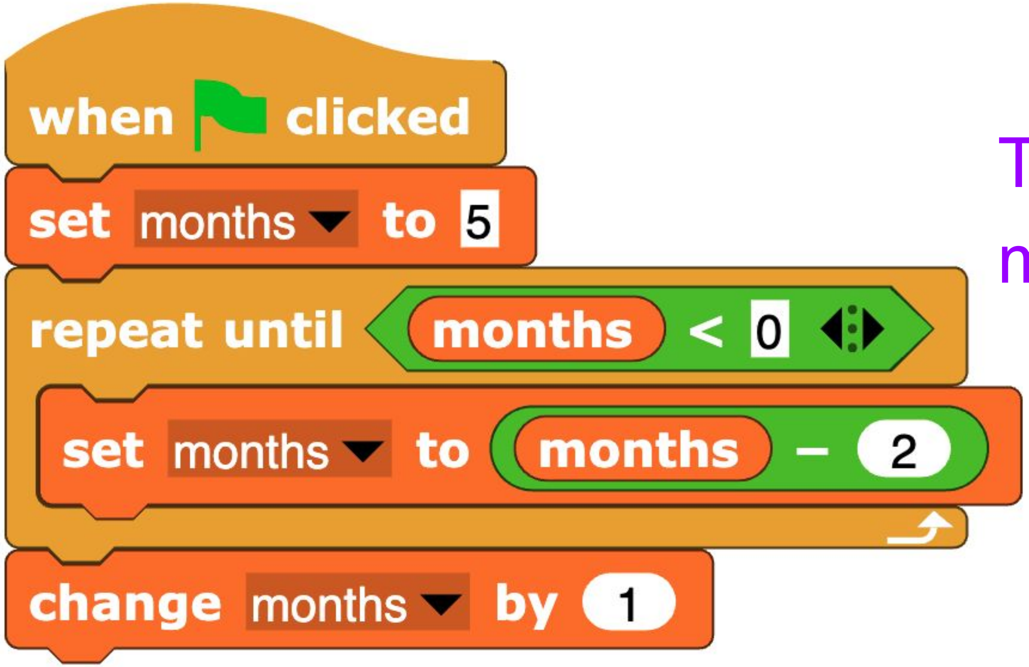
No → Execute code inside the loop.

Yes → Exit the loop, execute the first line after the loop.

Activity - Step through this code

Step 2.5: Go through each step of the loop, keep track of the variable and its value.

months 3



```
when [flag] clicked
set months to 5
repeat until <(months) < 0>
    set months to ((months) - 2)
change months by 1
```

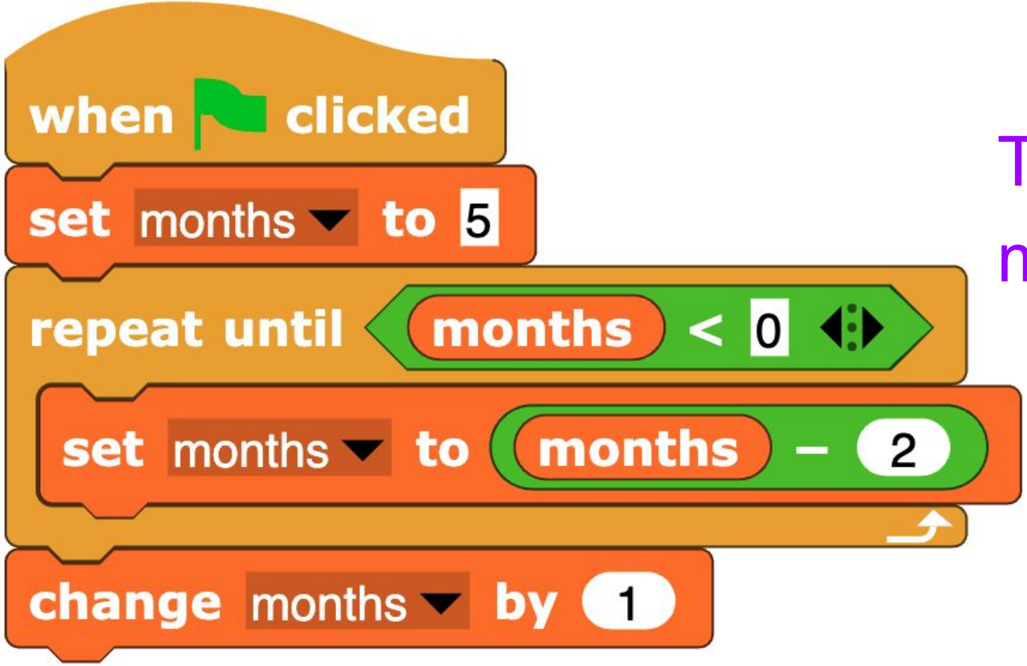
The value stored inside the months variable changes:

3 - 2 = 1

Activity - Step through this code

Step 2.5: Go through each step of the loop, keep track of the variable and its value.

months 1



```
when 🟩 clicked
set months to 5
repeat until < (months) < 0 >
    set months to ((months) − 2)
change months by 1
```

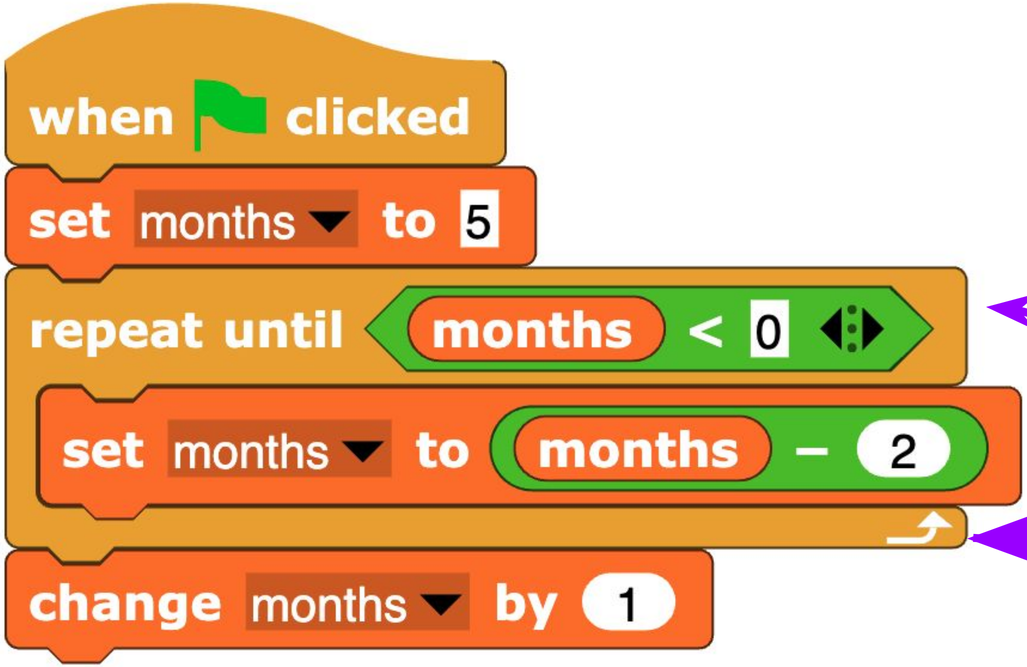
The value stored inside the months variable changes:

3 - 2 = 1

Activity - Step through this code

Step 2.6: Go through each step of the loop, keep track of the variable and its value.

months 1



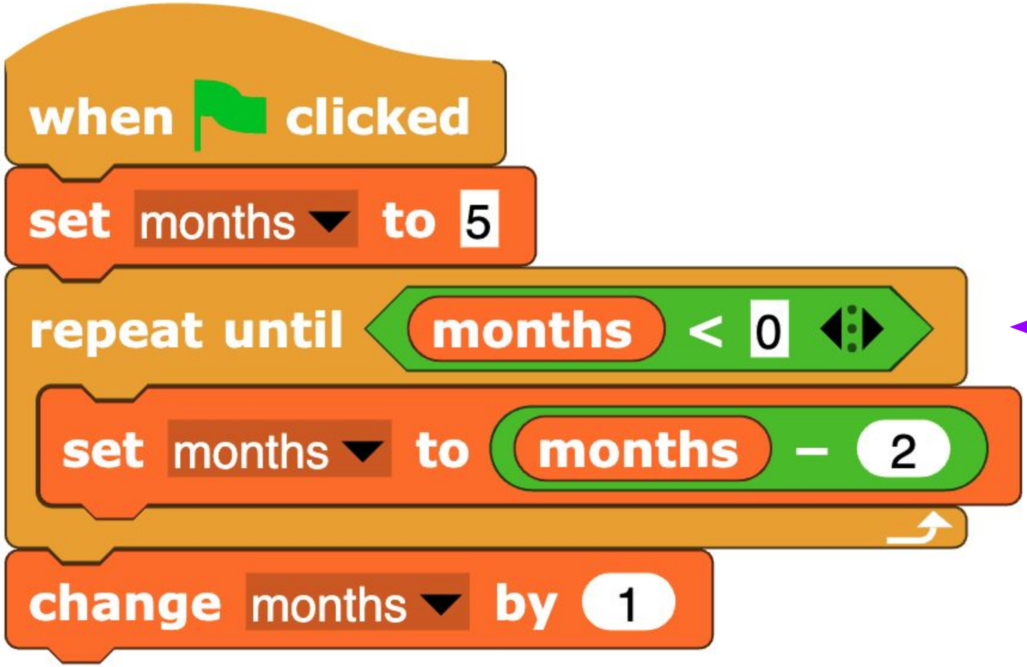
```
when [flag] clicked
set months to 5
repeat until < months < 0 >
    set months to months − 2
change months by 1
```

No more code inside the loop. We head back up to the start of the loop.

Activity - Step through this code

Step 2.7: Go through each step of the loop, keep track of the variable and its value.

months 1



```
when 🏳 clicked
set months to 5
repeat until months < 0
    set months to months − 2
change months by 1
```

Is the condition true?

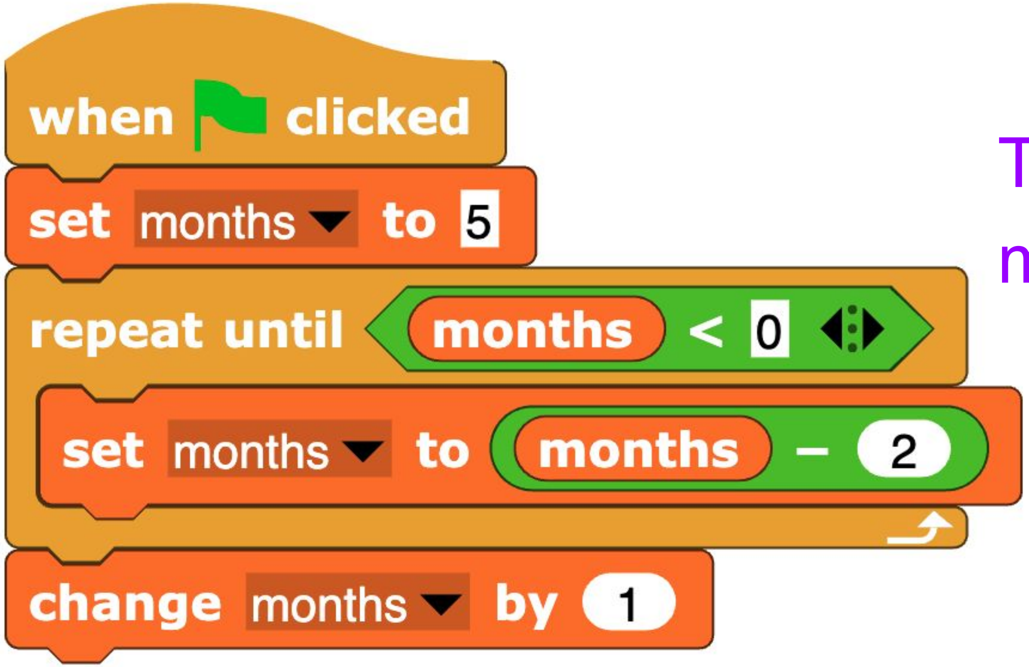
No → Execute code inside the loop.

Yes → Exit the loop, execute the first line after the loop.

Activity - Step through this code

Step 2.8: Go through each step of the loop, keep track of the variable and its value.

months 1



```
when [flag] clicked
set months to 5
repeat until (months < 0)
    set months to (months − 2)
change months by 1
```

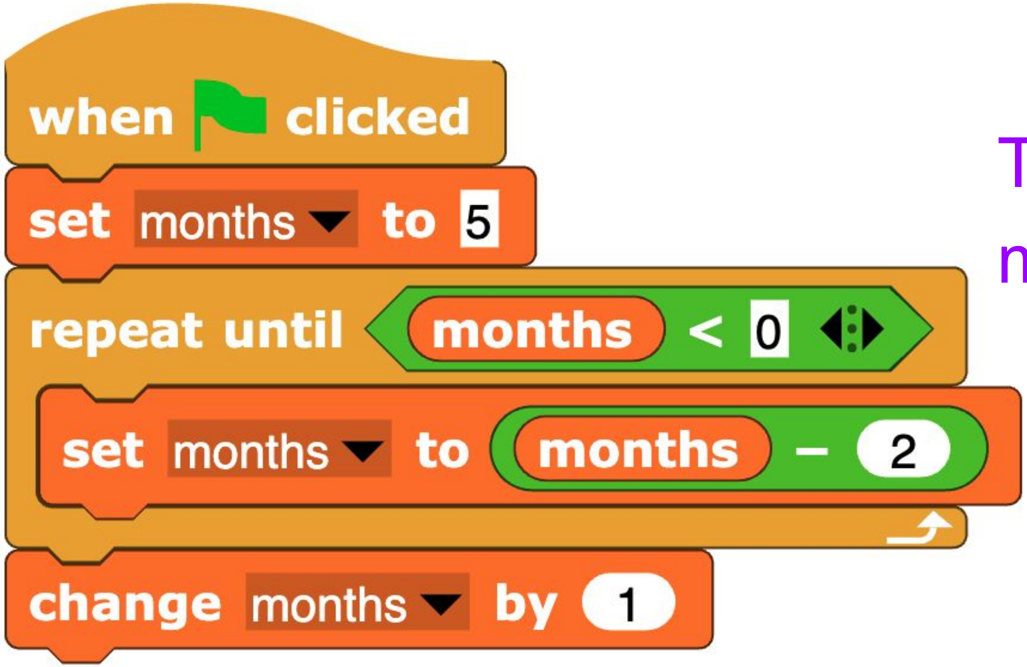
The value stored inside the months variable changes:

1 - 2 = -1

Activity - Step through this code

Step 2.8: Go through each step of the loop, keep track of the variable and its value.

months -1



```
when [flag] clicked
set months to 5
repeat until months < 0
    set months to months − 2
change months by 1
```

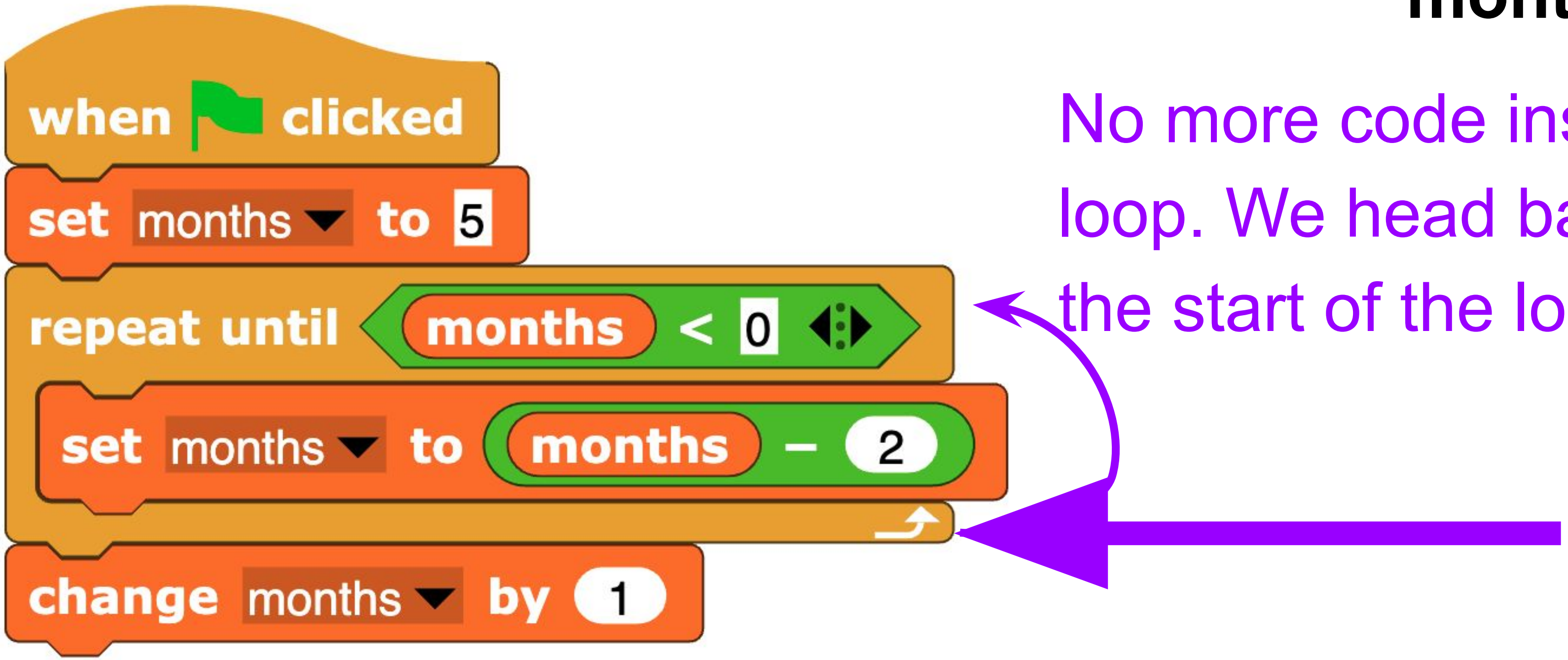
The value stored inside the months variable changes:

1 - 2 = -1

Activity - Step through this code

Step 2.9: Go through each step of the loop, keep track of the variable and its value.

months -1



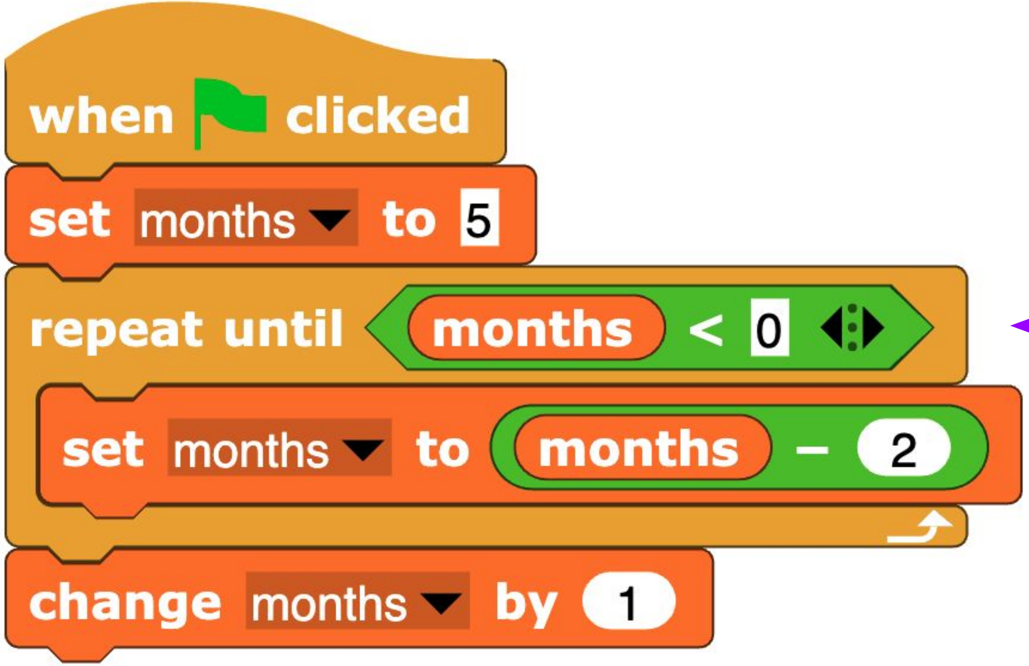
```
when [flag] clicked
set months to 5
repeat until <(months) < 0>
    set months to ((months) - 2)
change months by 1
```

No more code inside the loop. We head back up to the start of the loop.

Activity - Step through this code

Step 3.0: Write down the value of each variable after exiting the loop/repeat block.

months -1



```
when [flag] clicked
set months to 5
repeat until (months < 0)
    set months to (months - 2)
change months by 1
```

Is the condition true?

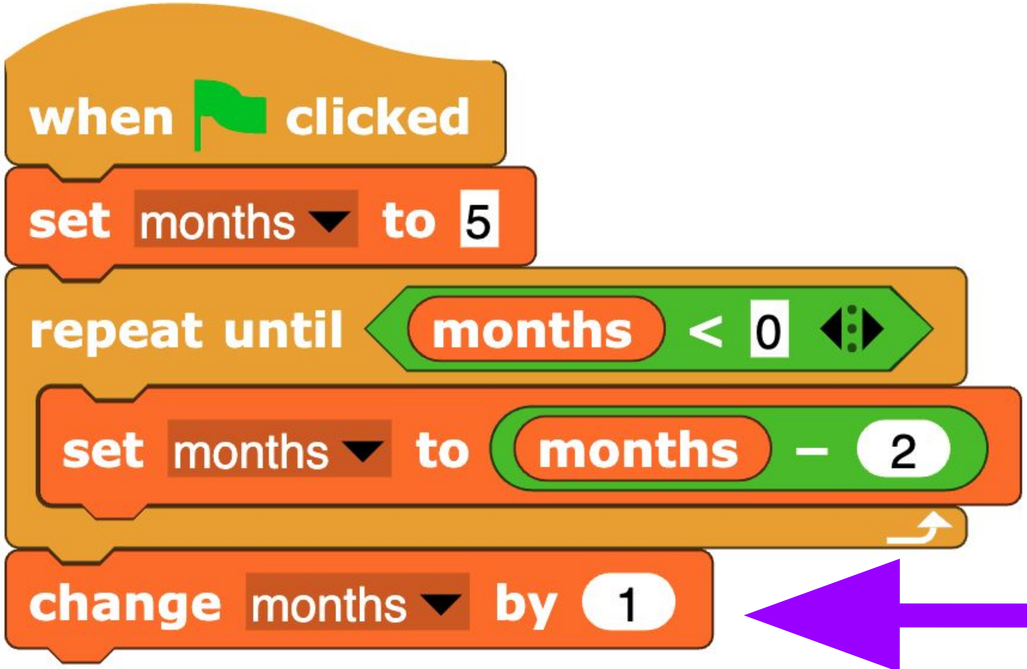
No → Execute code inside the loop.

Yes → Exit the loop, execute the firs line after the loop.

Activity - Step through this code

Step 3.1: Go through any remaining code, keep track of the variable and its value.

months 0



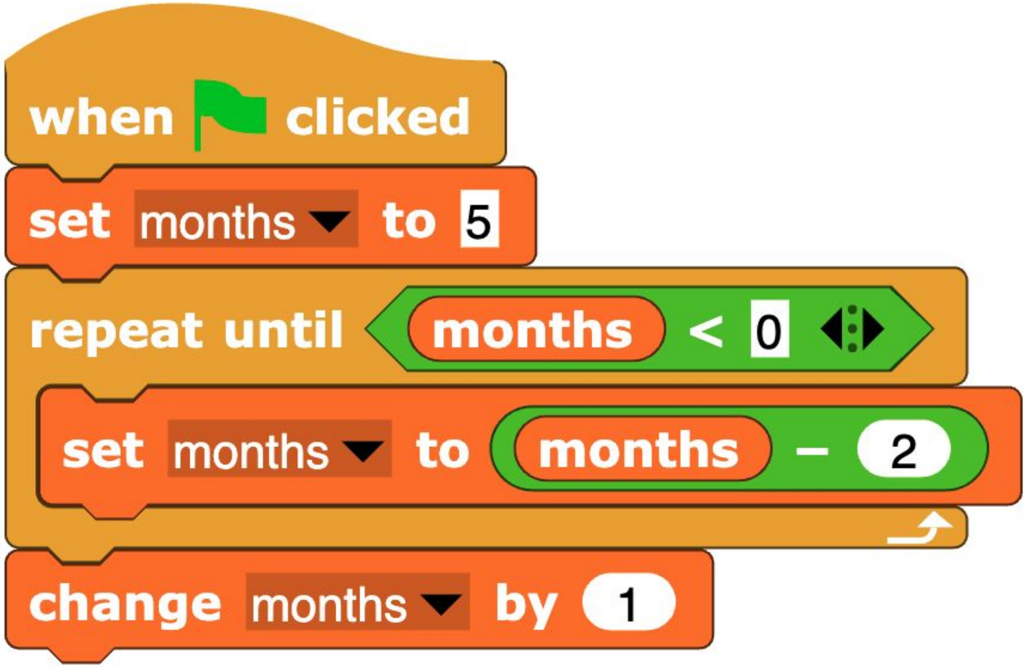
```
when ⚑ clicked
set months ▾ to 5
repeat until ( months < 0 )
    set months ▾ to ( months − 2 )
change months ▾ by 1
```

The value stored inside the months variable changes:

-1 + 1 = 0

Activity - Step through this code

Step 4: Celebrate! You have solved the question

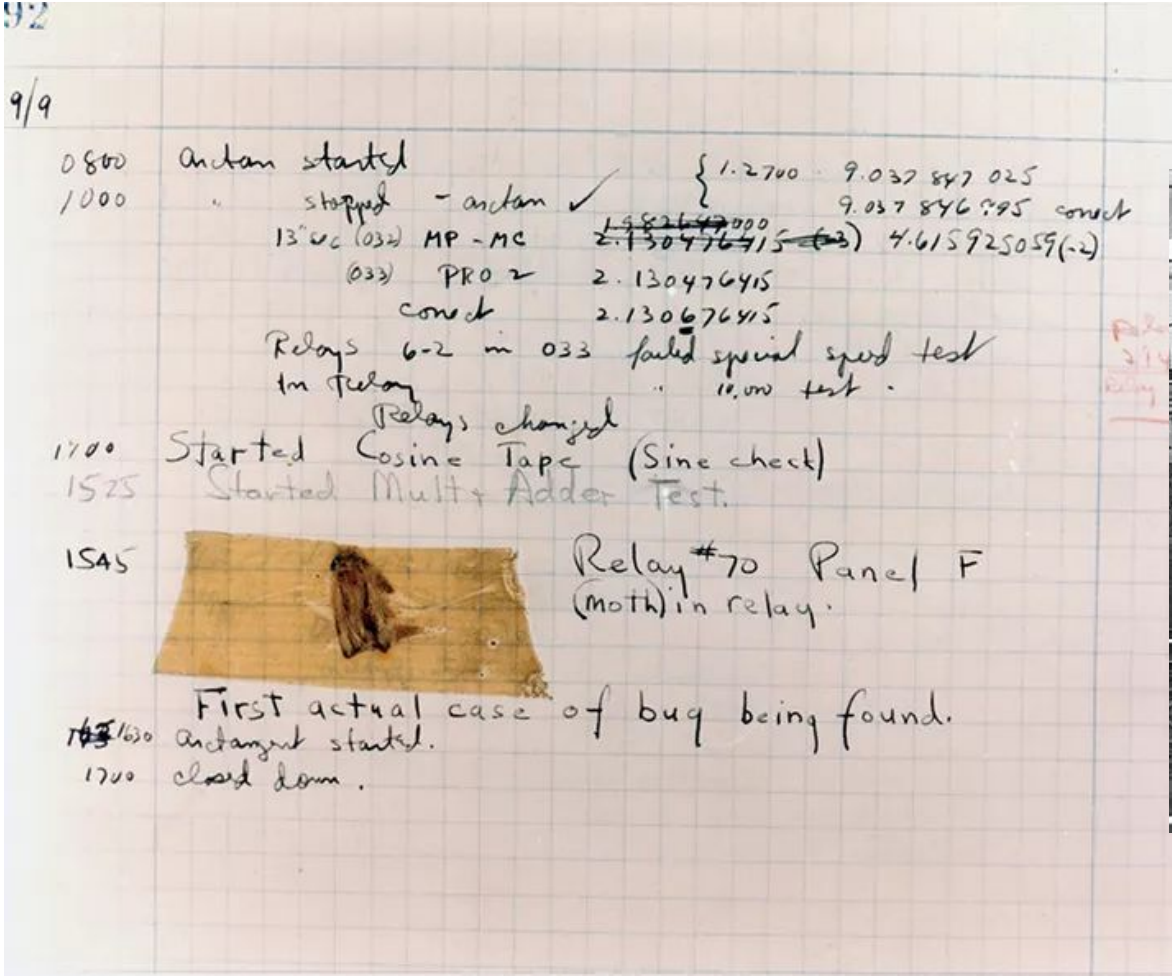


```
when [flag] clicked
set months to 5
repeat until <(months) < 0>
    set months to ((months) - 2)
change months by 1
```

months 0

Done!

Debugging



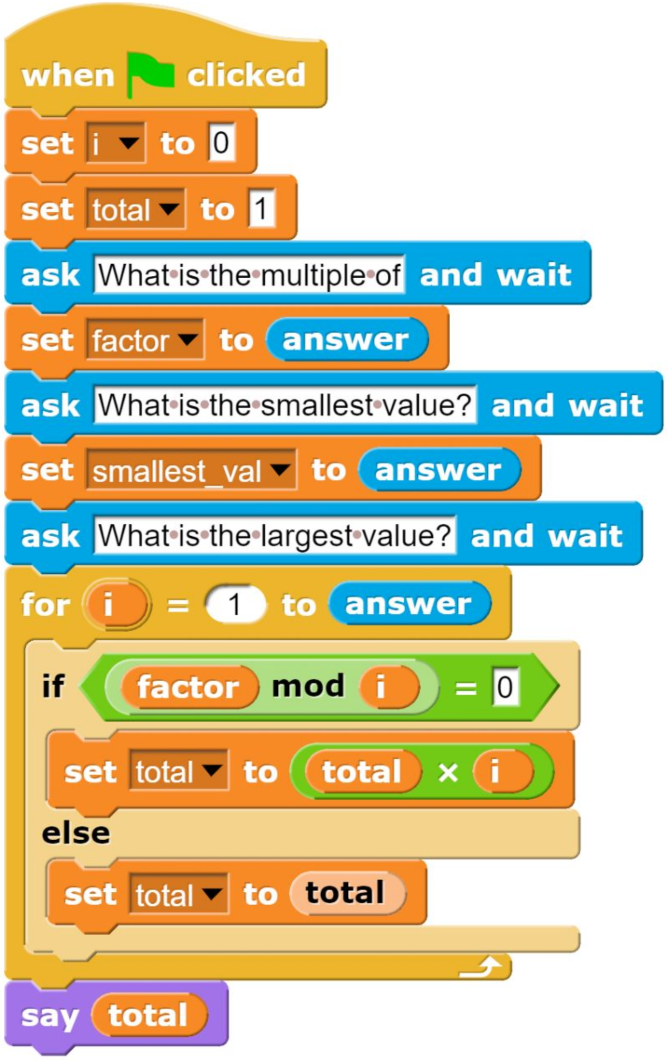
92

9/9

0800 Anctan started
1000 " stopped - anctan ✓ {1.2700 9.037 847 025
 9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ 2.130476415 ~~(-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In Relay " 10,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult+ Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~143~~ 1630 Antangent started.
1700 closed down.

Reference: The "bug" [Link]

Activity



```
when ⚑ clicked
set i to 0
set total to 1
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
set smallest_val to answer
ask What is the largest value? and wait
for i = 1 to answer
    if factor mod i = 0
        set total to total × i
    else
        set total to total
say total
```

This code block is supposed to find the product between two positive integers (not inclusive)

Example, if the user inputs:

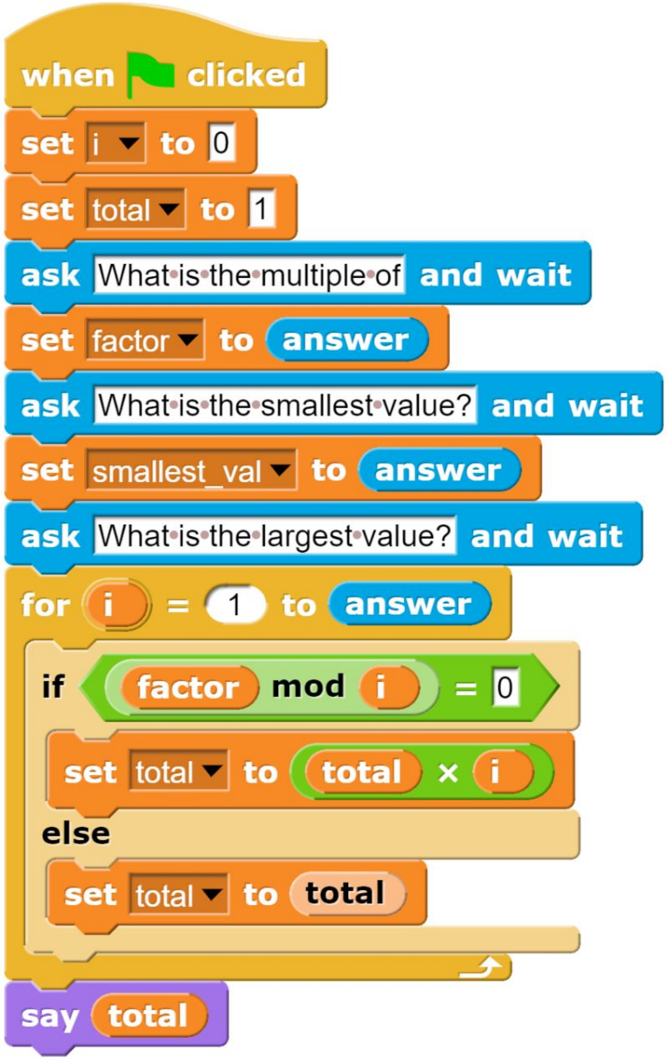
What is the multiple of? **2**

What is the smallest value? **1**

What is the largest value? **10**

The result should be:

2x4x6x8 = 384



```
when ⚑ clicked
set i to 0
set total to 1
ask What is the multiple of and wait
set factor to answer
ask What is the smallest value? and wait
set smallest_val to answer
ask What is the largest value? and wait
for i = 1 to answer
    if (factor mod i) = 0
        set total to (total × i)
    else
        set total to total
say total
```

Review the code block and identify any bug(s).

1. Clearly highlight the problematic code [bug(s)]
2. Explain in plain English what needs to be changed so the code works properly

That's it!

That's all the programming basics you need to know…(for now).

Programming is a LOT easier to learn by *doing* than by *watching*!

Modulo Operator

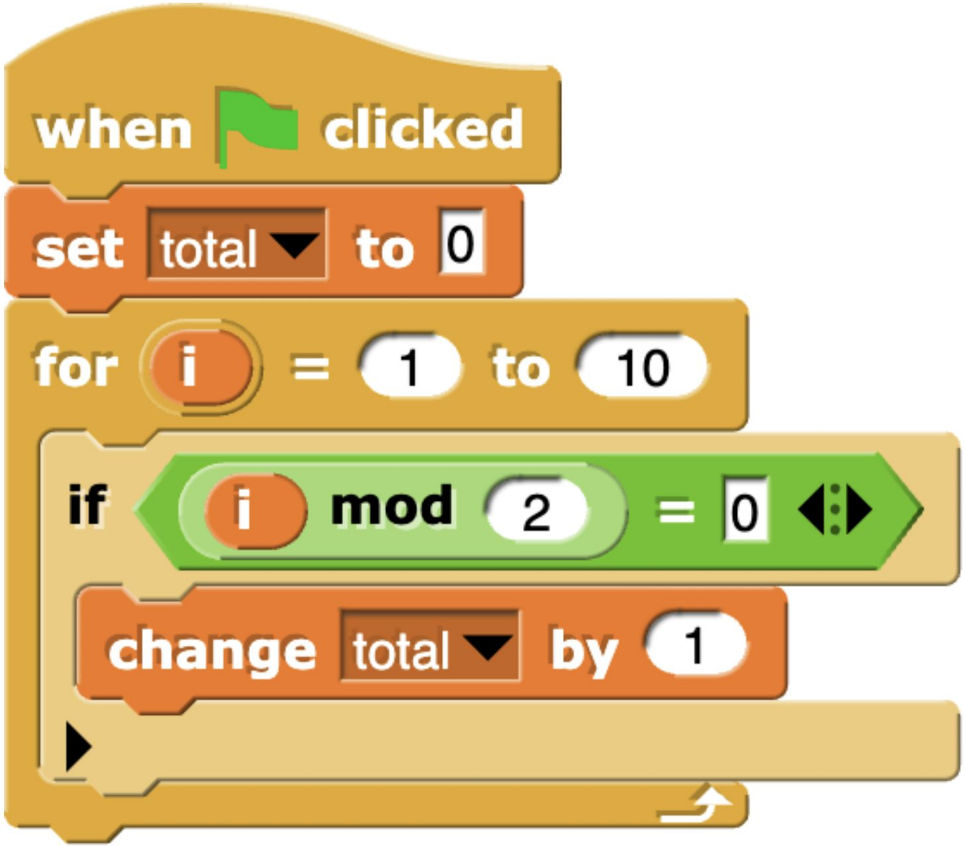
Learning Goals

After this today's lecture, you should be able to:

- Understand and explain the modulo (mod) operator.
- Apply mod operator in Snap! programming
- Understand the history & importance of debugging in programming
- Identify any bugs associated with a given code block
- Explain in plain English what needs to be changed to resolve bugs
- *Bonus: understand AM/PM acronym in the clock system*

Q: What is the value of total when this code block is run?

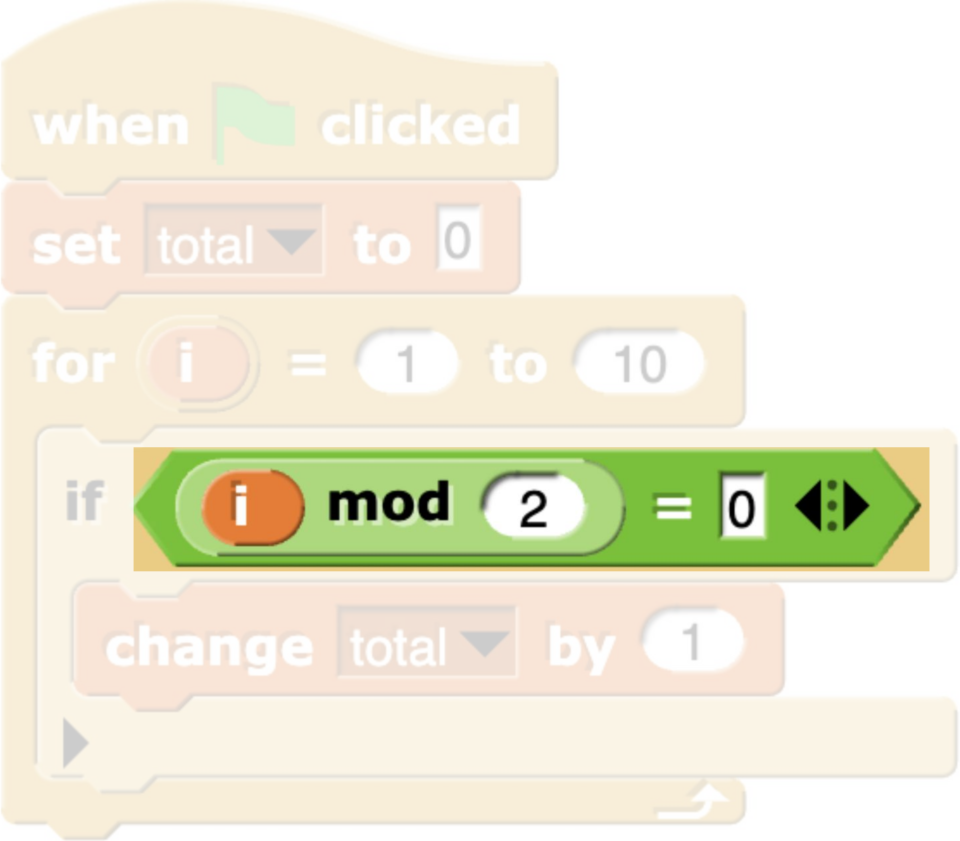
What does this code block do?



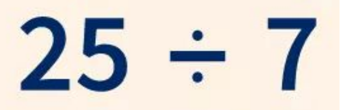
```
when 🏴 clicked
set total to 0
for i = 1 to 10
    if i mod 2 = 0
        change total by 1
```

"Mod" Operator

Short for *modulo*



when clicked
set total to 0
for i = 1 to 10
    if i mod 2 = 0
        change total by 1



$25 \div 7$

Image Source

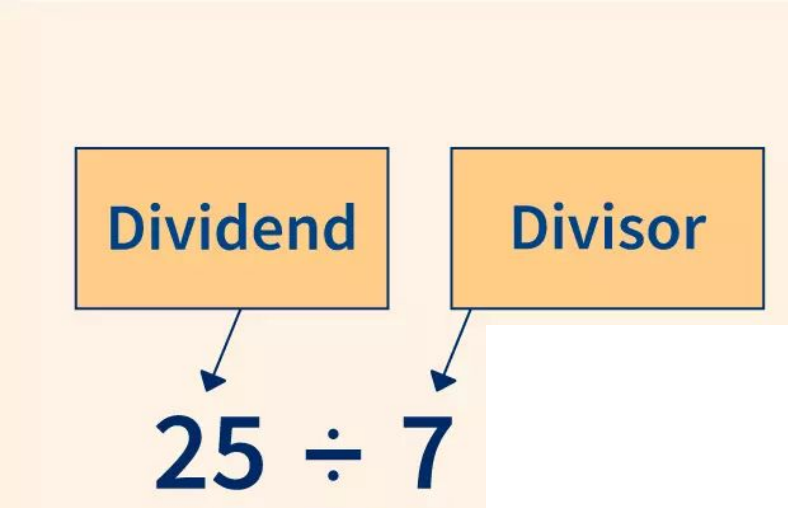


Image Source

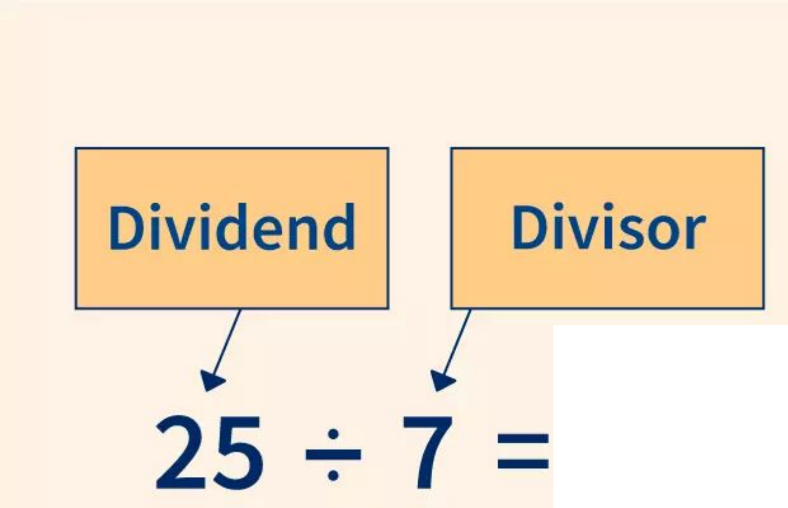


Image Source

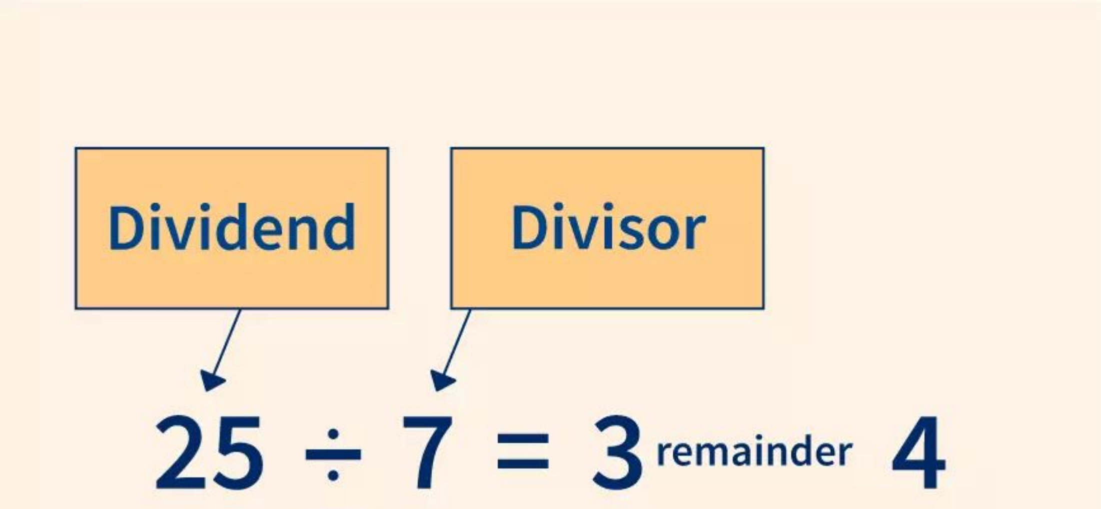
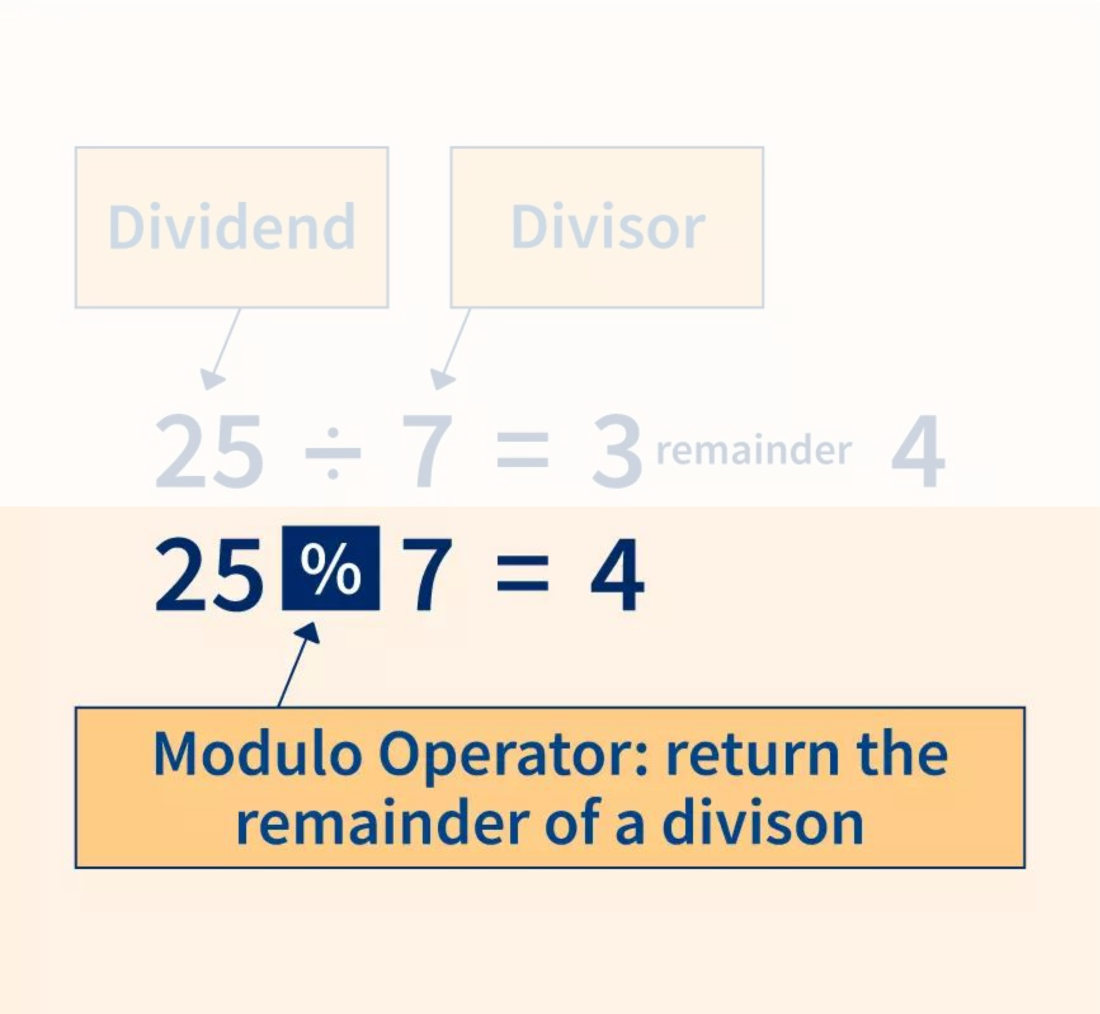


Image Source



Dividend

Divisor

25 ÷ 7 = 3 remainder 4

25 % 7 = 4

Modulo Operator: return the remainder of a divison

Image Source

Examples

- Clock System (AM/PM)
  - Before noon: **A**nte **M**eridiem (AM) → 12:01am to 11:59am
  - After noon: **P**ost **M**eridiem (PM) → 12:00pm to 11:59pm

Examples

- Clock System (AM/PM)
  - Before noon: **A**nte **M**eridiem (AM) → 12:01am to 11:59am
  - After noon: **P**ost **M**eridiem (PM) → 12:00pm to 11:59pm
- Clock System (military)
  - 24 hour system → after 12: we keep on counting (13, 14..)

Examples

- Clock System (AM/PM)
  - Before noon: **A**nte **M**eridiem (AM) → 12:01am to 11:59am
  - After noon: **P**ost **M**eridiem (PM) → 12:00pm to 11:59pm
- Clock System (military)
  - 24 hour system → after 12: we keep on counting (13, 14..)
- To convert between these two, we use a **mod operator!**
  - Our class starts at 15:00 → 3pm
  - 15 **mod** 12 = 3 *(since dividing 15 by 12, the remainder is 3)*

More Examples

- 5 mod 2 = 1 (the closest divisor is [2], 2x2 = **4**, the remainder is 1)
- 9 mod 3 = 0 (since 9 is exactly divisible by 3 with **no** remainder)
- 17 mod 5 = 2 (the closest divisor is [3], 5x3 = **15**, the remainder is 2)

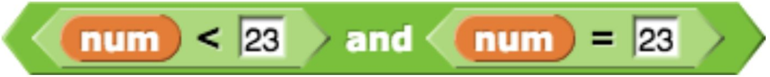
Wrap up

Take-Home Practice

Q: There’s no ≤ block in Snap! Suppose we wanted to build one. Which of the following Boolean expressions is equivalent to the expression num ≤ 23 ?

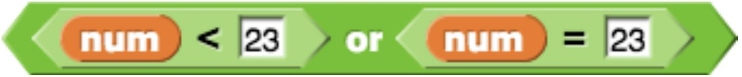
iClicker

A



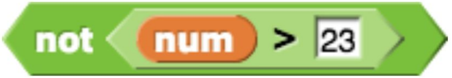
num < 23 and num = 23

B



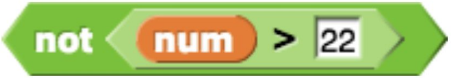
num < 23 or num = 23

C

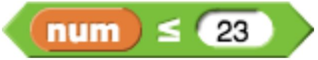


not num > 23

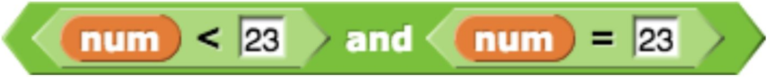
D



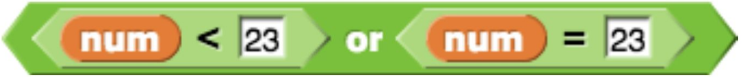
not num > 22

Q: There's no ≤ block in Snap! Suppose we wanted to build one. Which of the following Boolean expressions is equivalent to the expression  num ≤ 23 ?

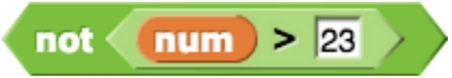
iClicker

A 

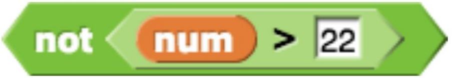
```
num < 23 and num = 23
```

[✓ B] 

```
num < 23 or num = 23
```

[✓ C] 

```
not num > 23
```

D 

```
not num > 22
```

Q: What is the value in total when the code is run, assuming user input = 3?

iClicker

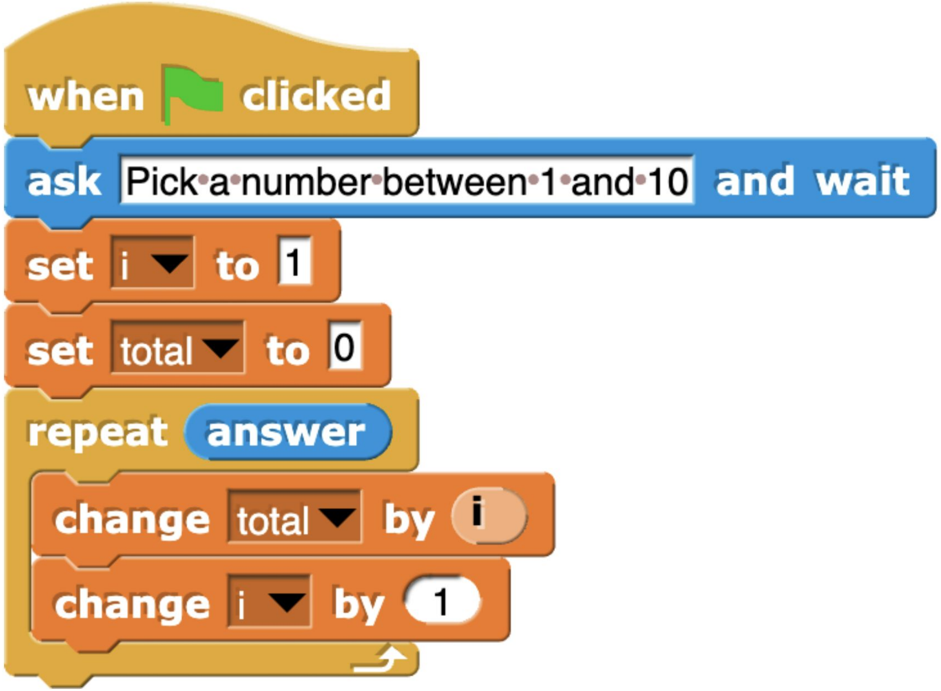
A. 2

B. 3

C. 4

D. 6

E. 10



```
when 🏴 clicked
ask Pick a number between 1 and 10 and wait
set i ▼ to 1
set total ▼ to 0
repeat answer
    change total ▼ by i
    change i ▼ by 1
```

Q: What is the value in total when the code is run, assuming user input = 3?

iClicker

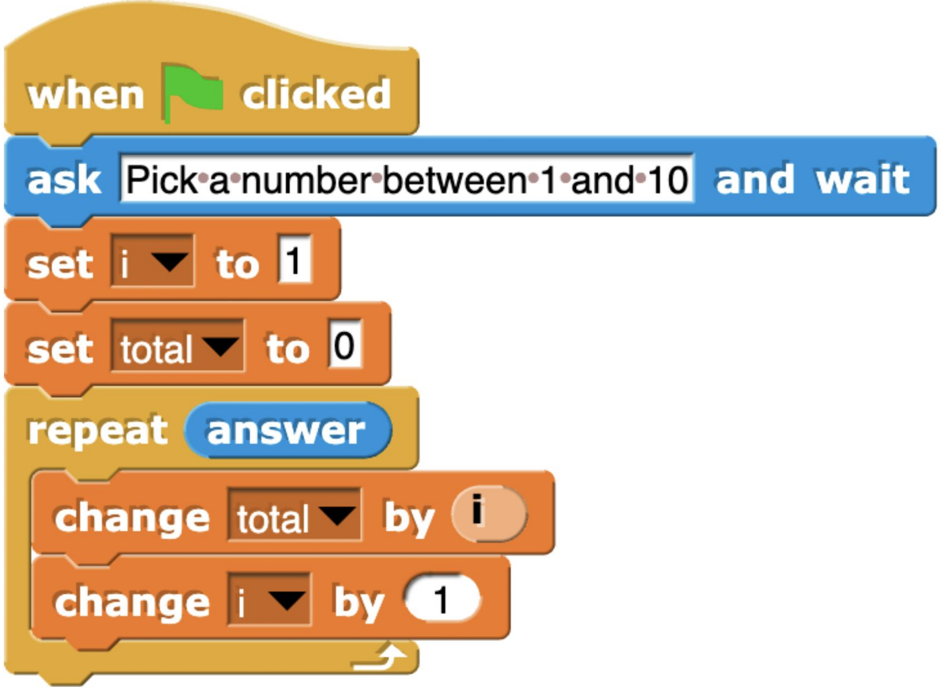
A. 2

B. 3

C. 4

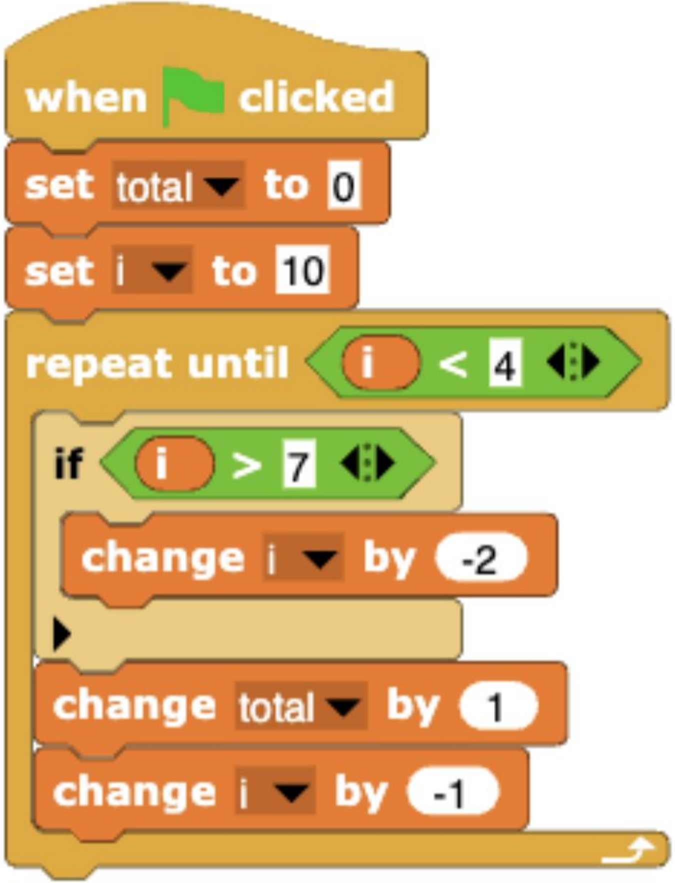
[✓ D. 6]

E. 10



```
when [flag] clicked
ask Pick a number between 1 and 10 and wait
set i to 1
set total to 0
repeat answer
    change total by i
    change i by 1
```

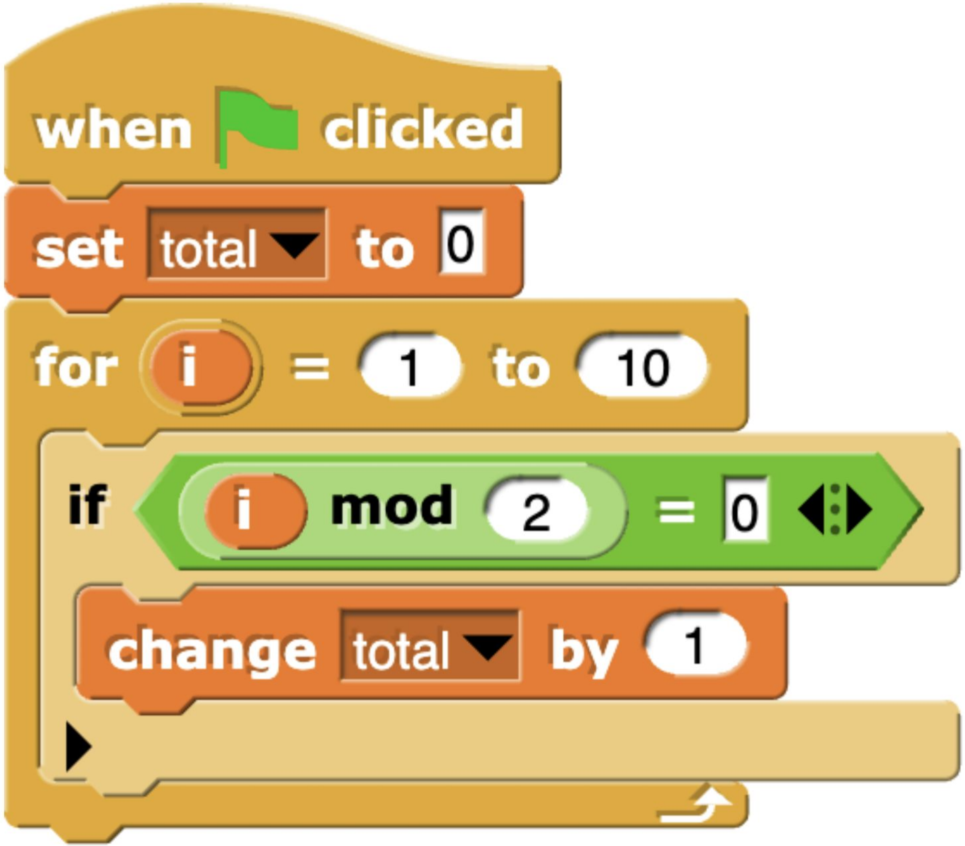
Q: What is the value of total and i when this code block is run?



```
when 🏳 clicked
set total to 0
set i to 10
repeat until (i < 4)
    if (i > 7)
        change i by -2
    change total by 1
    change i by -1
```

Q: What is the value of total when this code block is run?

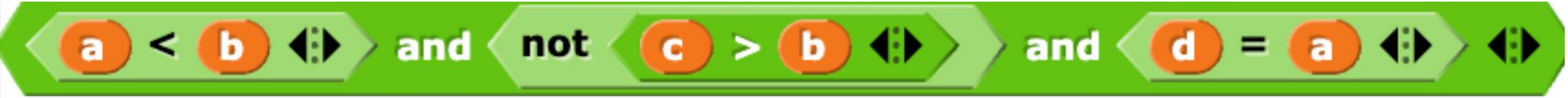
What does this code block do?



```
when ⚑ clicked
set total ▼ to 0
for i = 1 to 10
    if i mod 2 = 0
        change total ▼ by 1
```

Q: What will the following expression evaluate to based on the given values?

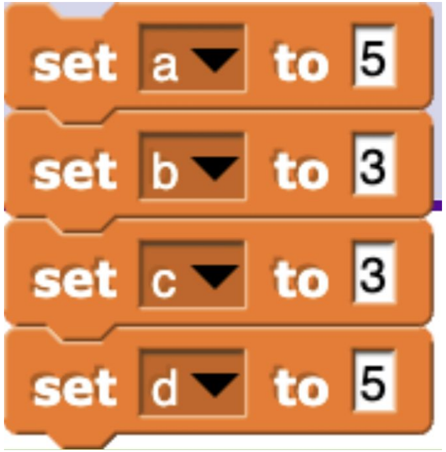
iClicker



```
a < b and not c > b and d = a
```

A. True

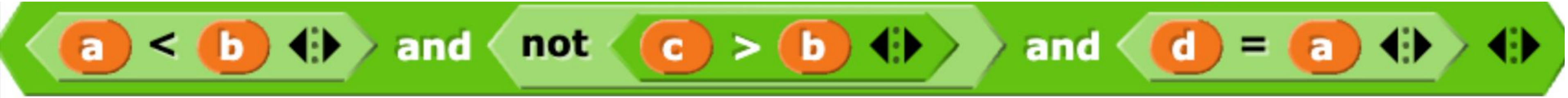
B. False



```
set a to 5
set b to 3
set c to 3
set d to 5
```

Q: What will the following expression evaluate to based on the given values?

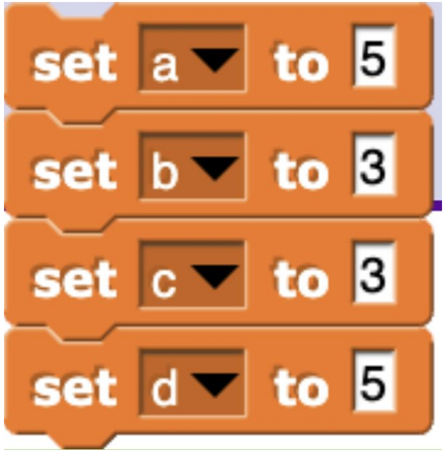
iClicker



```
(a < b) and (not (c > b)) and (d = a)
```

A. True

[✓ B. False]



```
set a to 5
set b to 3
set c to 3
set d to 5
```

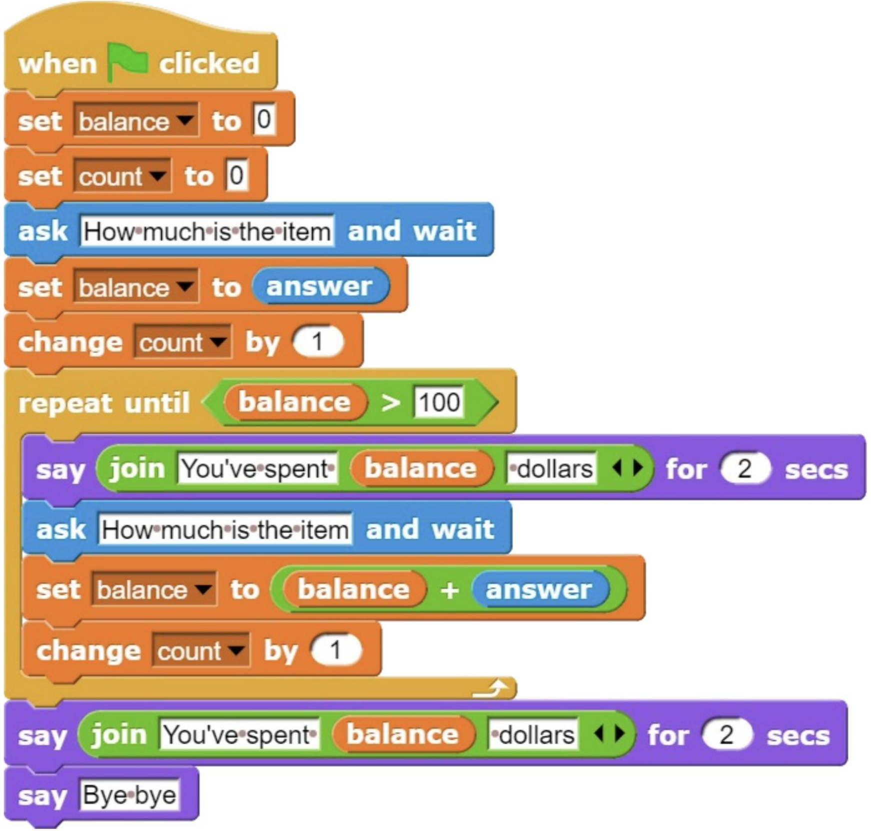
What does this code block do?

Describe the code in terms of input and output and what is being done.

The block consumes....

It does...

It produces/displays/reports…

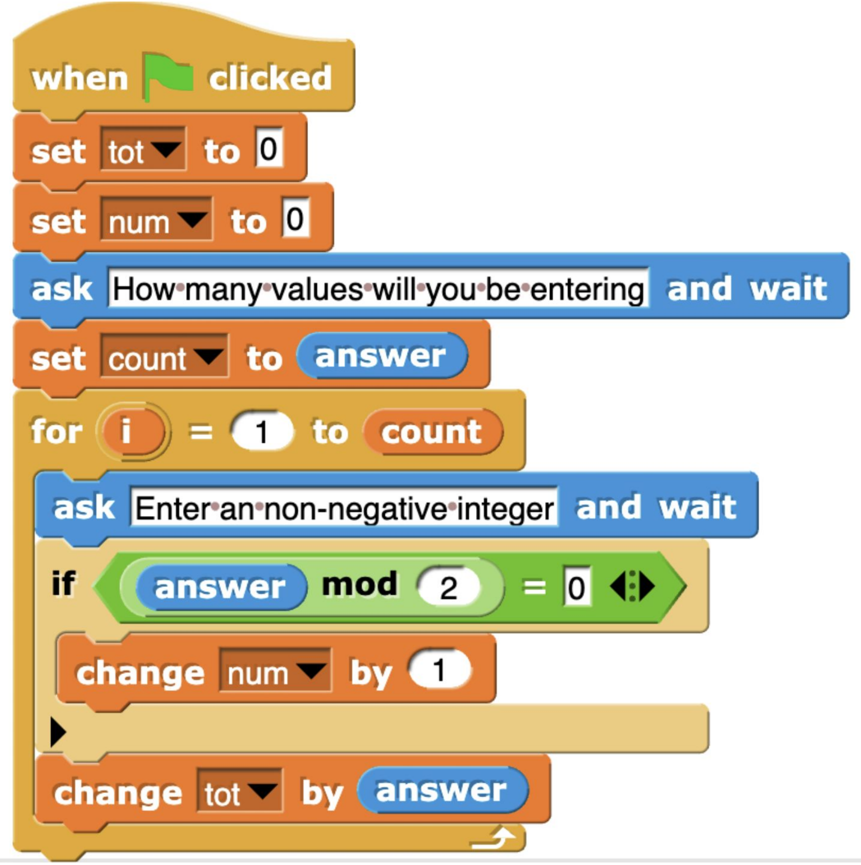


```
when (flag) clicked
set balance to 0
set count to 0
ask How much is the item and wait
set balance to answer
change count by 1
repeat until balance > 100
    say join You've spent balance dollars for 2 secs
    ask How much is the item and wait
    set balance to balance + answer
    change count by 1
say join You've spent balance dollars for 2 secs
say Bye bye
```

What does this code block do?

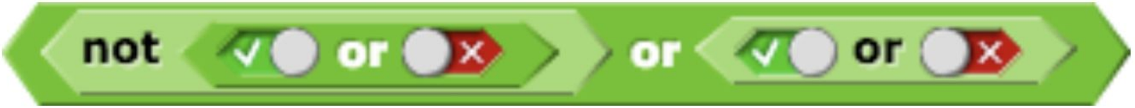
What is the value of **tot**, **num** and **count** when the block is run? *[for practice, you can assume a number between 1-6]*

What does this code block do?



```
when (flag) clicked
set tot to 0
set num to 0
ask How many values will you be entering and wait
set count to answer
for i = 1 to count
    ask Enter an non-negative integer and wait
    if answer mod 2 = 0
        change num by 1
    change tot by answer
```

Q: What will the following expression report?

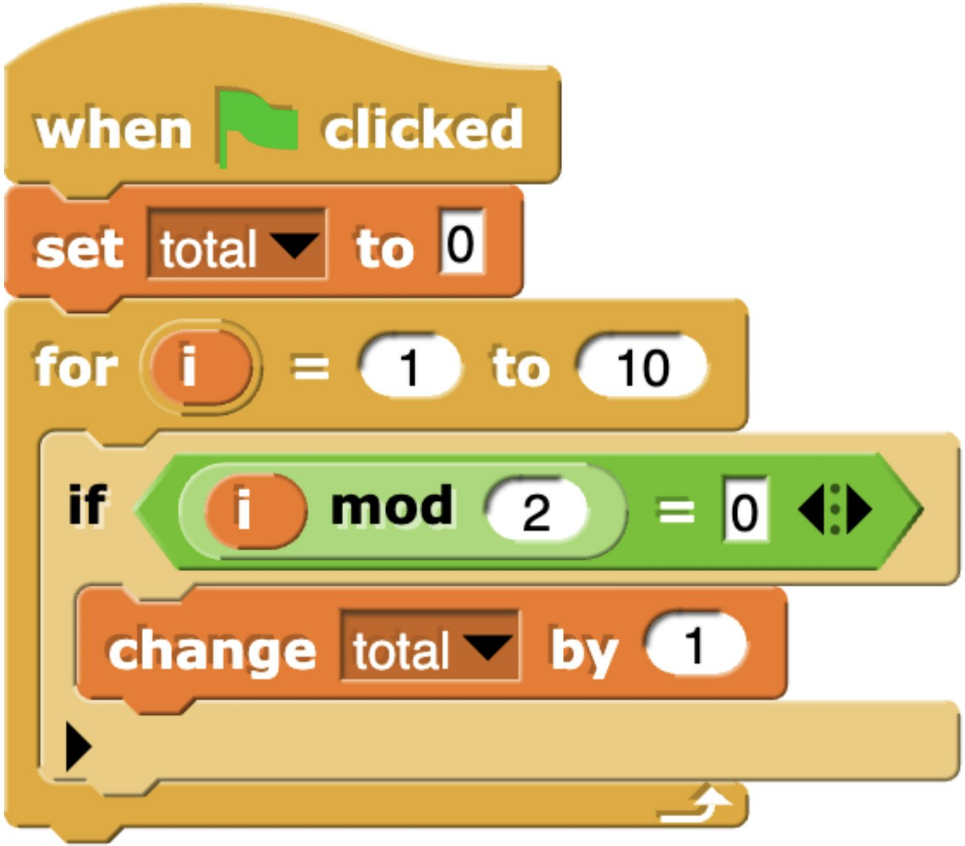


A. True

B. False

Q: What is the value of total when this code block is run?

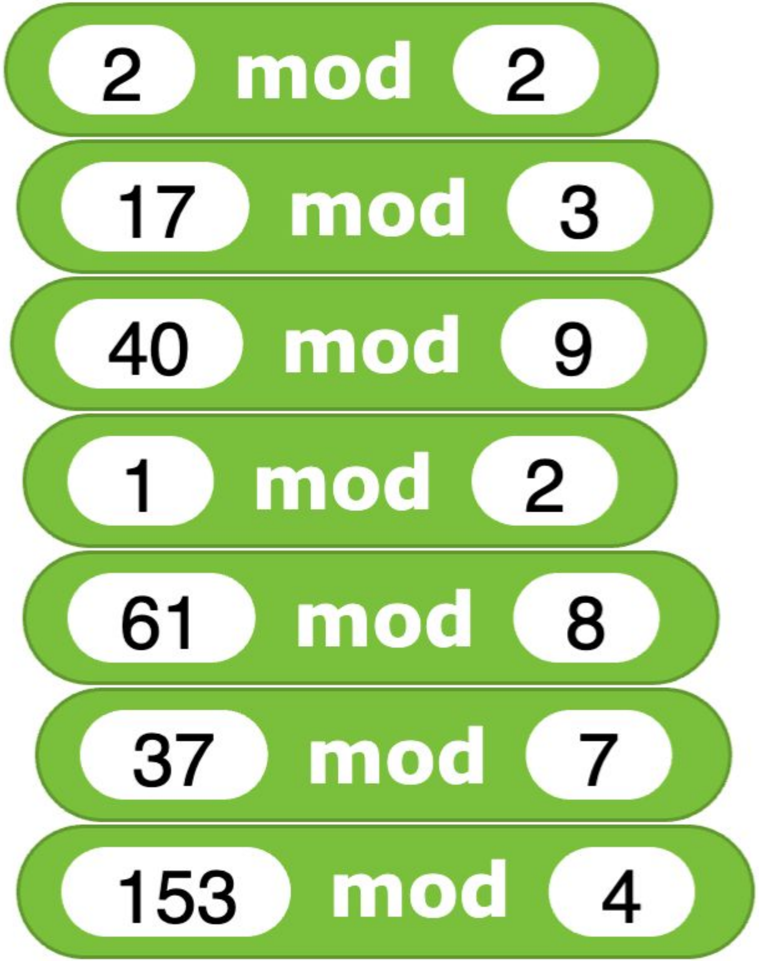
What does this code block do?



```
when [flag] clicked
set total to 0
for i = 1 to 10
    if i mod 2 = 0
        change total by 1
```

More Examples

- 5 mod 2 = 1 (the closest divisor is [2], 2x2 = **4**, the remainder is 1)
- 9 mod 3 = 0 (since 9 is exactly divisible by 3 with **no** remainder)
- 17 mod 5 = 2 (the closest divisor is [3], 5x3 = **15**, the remainder is 2)
- 25 mod 3 = 1 (the closest divisor is [8], 3x8 = **24**, the remainder is 1)
- 44 mod 10 = 4 (the closest divisor is [4], 10x4 = **40**, the remainder is 4)
- 53 mod 6 = 5 (the closest divisor is [8], 6x8 = **48**, the remainder is 5)
- 72 mod 8 = 0 (since 72 is exactly divisible by 8 with **no** remainder)



2 mod 2 =

17 mod 3 =

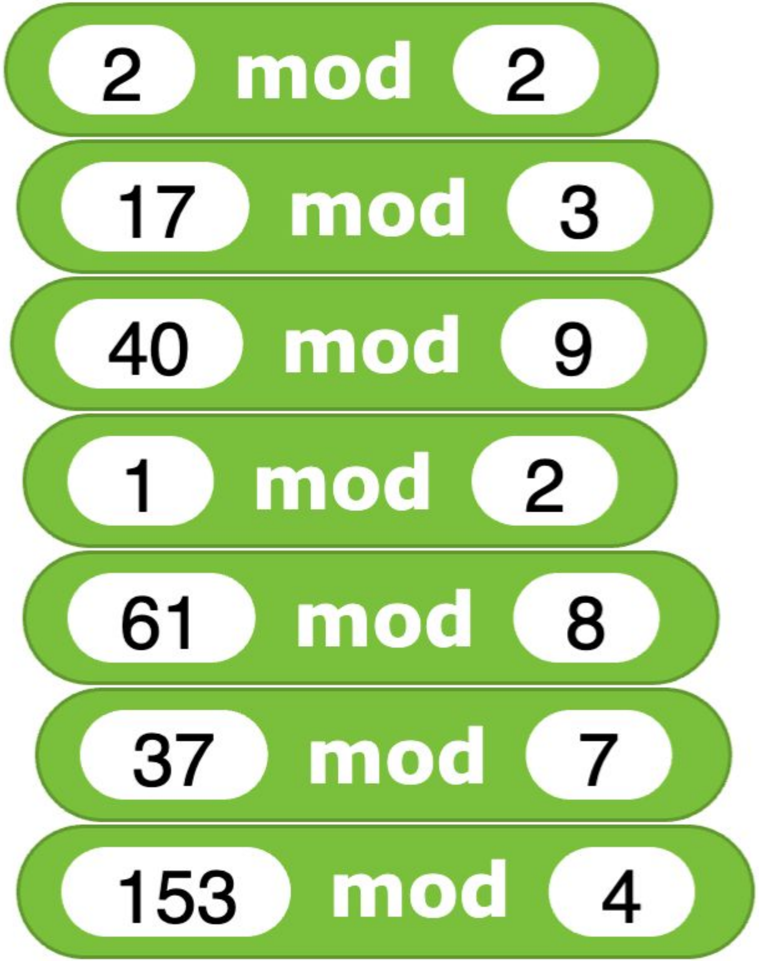
40 mod 9 =

1 mod 2 =

61 mod 8 =

37 mod 7 =

153 mod 4 =



2 mod 2 = 0

17 mod 3 = 2

40 mod 9 = 4

1 mod 2 = 1

61 mod 8 = 5

37 mod 7 = 2

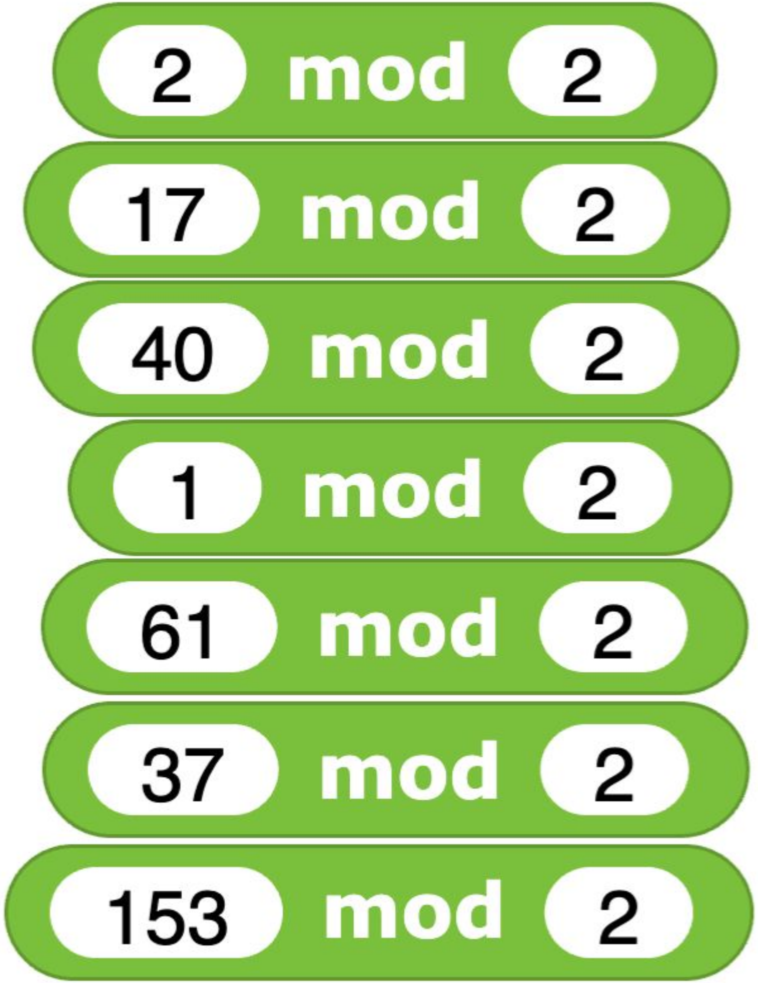
153 mod 4 = 1

Programming Context:

- This operator is helpful in programming to check **if a number is even or odd**, **looping through a range of values**, and **creating patterns**.

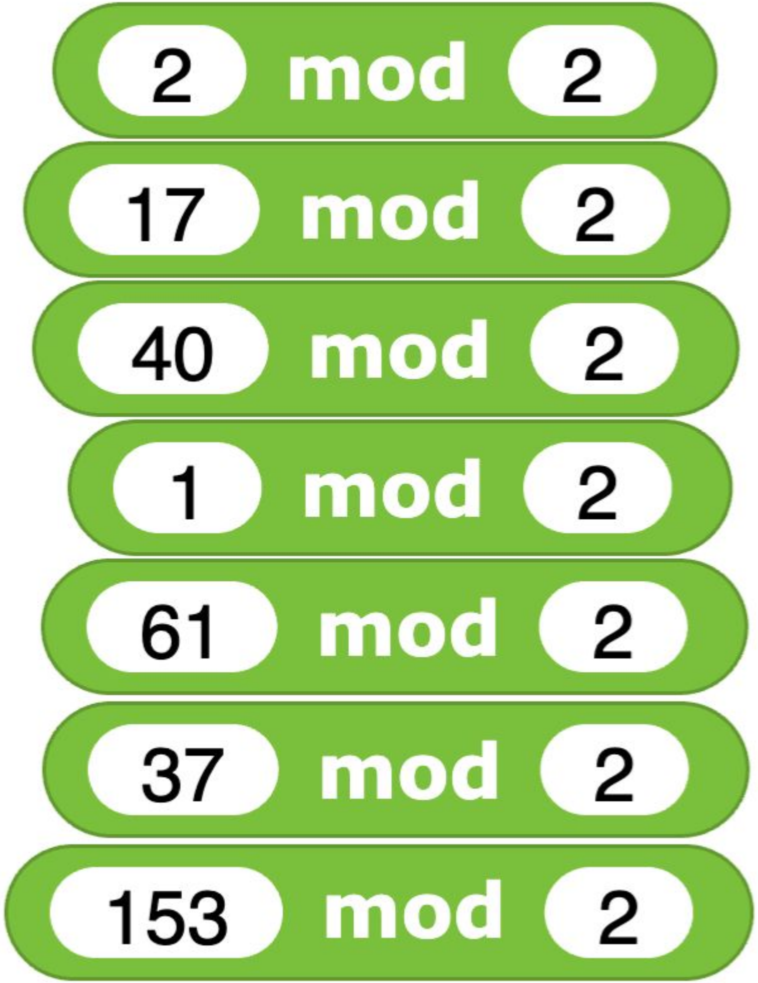
Programming Context:

- This operator is helpful in programming to check **if a number is even or odd**, **looping through a range of values**, and **creating patterns**.
- An even number will have a remainder of 0 **<u>when divided by 2</u>**, while an odd number will have a remainder of 1
  - 7 mod 2 = 1 (Odd)
  - 12 mod 2 = 0 (Even)



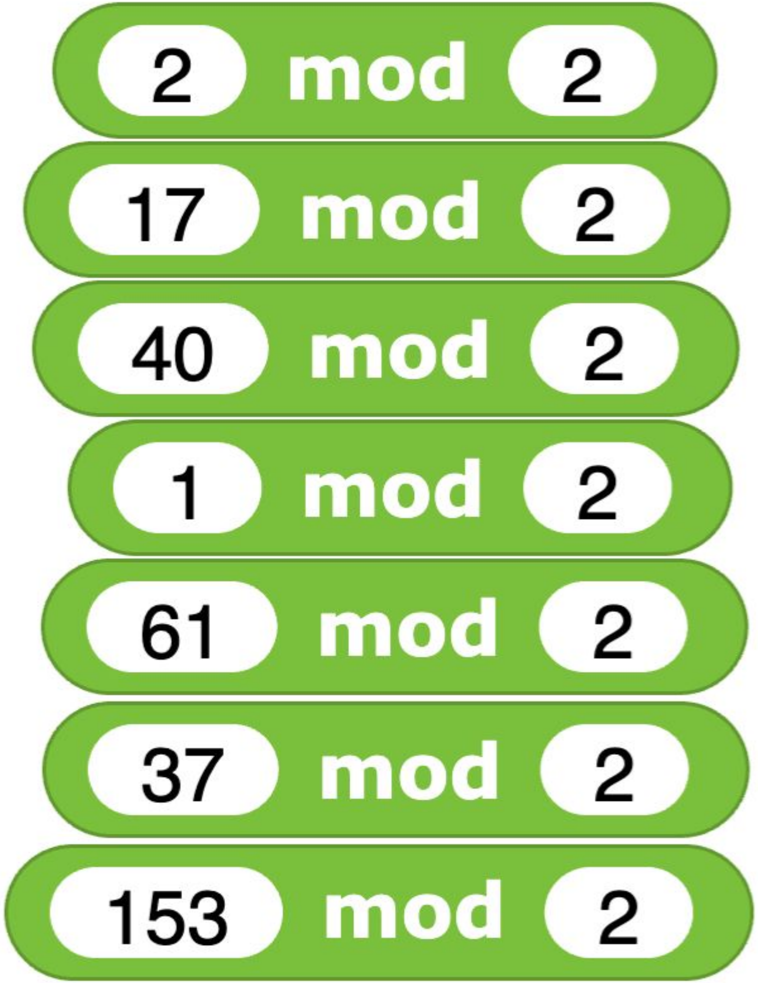
```
2 mod 2
17 mod 2
40 mod 2
1 mod 2
61 mod 2
37 mod 2
153 mod 2
```

What do these Arithmetic Operators evaluate to? Odd or even?



```
2 mod 2
17 mod 2
40 mod 2
1 mod 2
61 mod 2
37 mod 2
153 mod 2
```

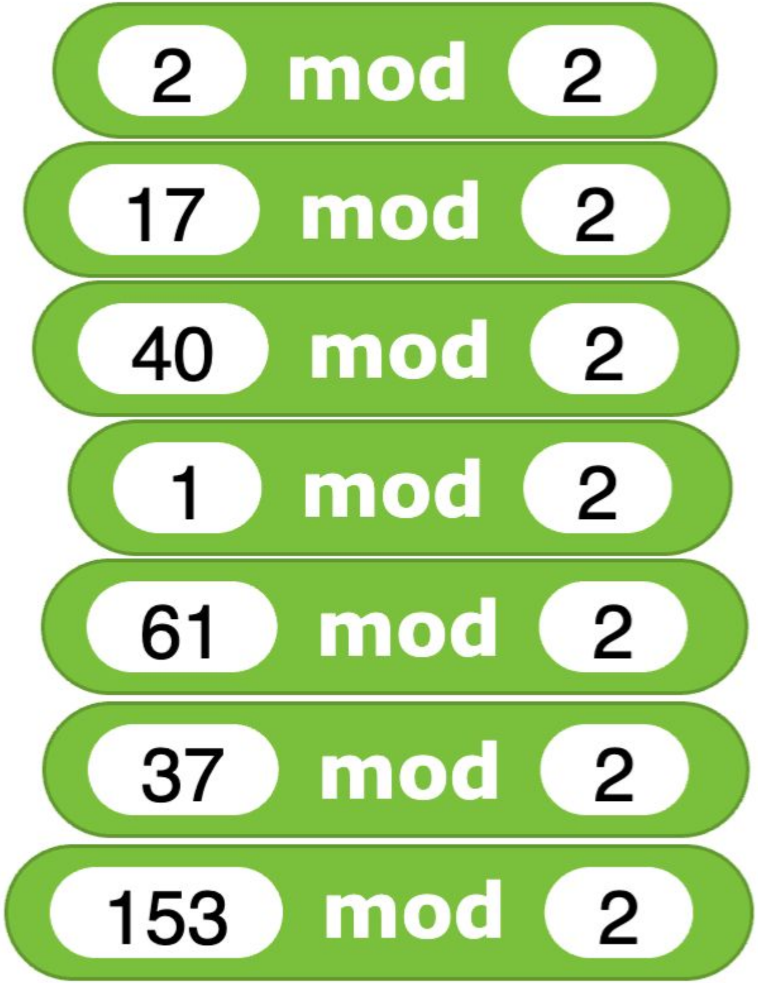
= 0 (even)



```
2 mod 2
17 mod 2
40 mod 2
1 mod 2
61 mod 2
37 mod 2
153 mod 2
```

= 0 (even)

= 1 (odd)

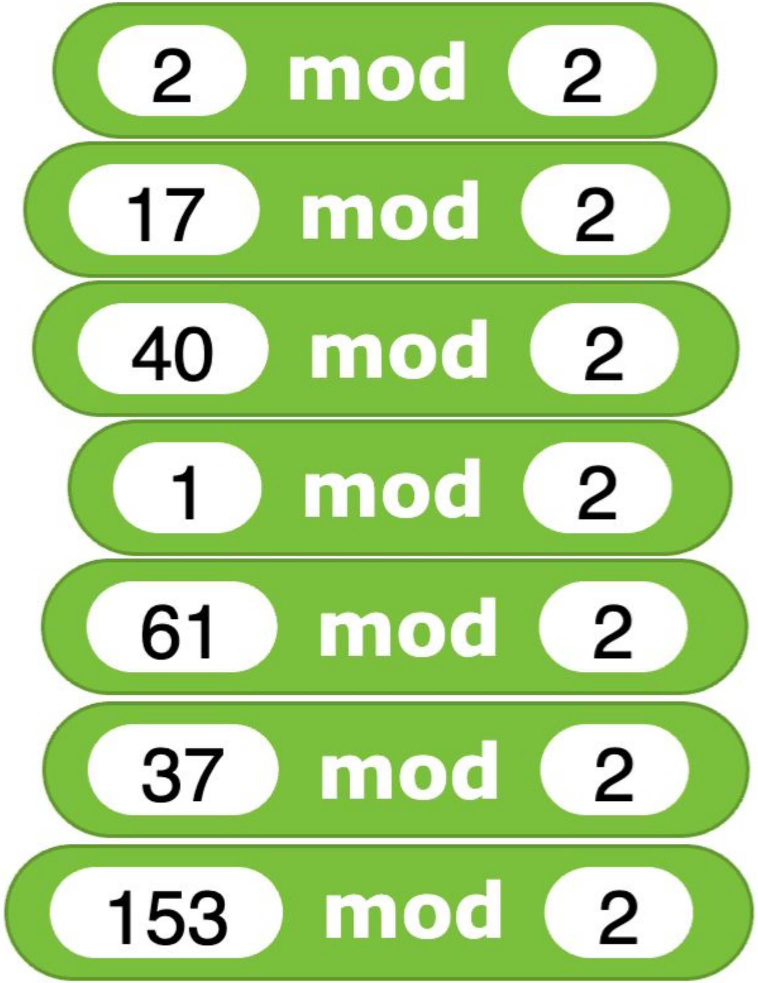


```
2 mod 2
17 mod 2
40 mod 2
1 mod 2
61 mod 2
37 mod 2
153 mod 2
```

= 0 (even)

= 1 (odd)

= 0 (even)



2 mod 2

17 mod 2

40 mod 2

1 mod 2

61 mod 2

37 mod 2

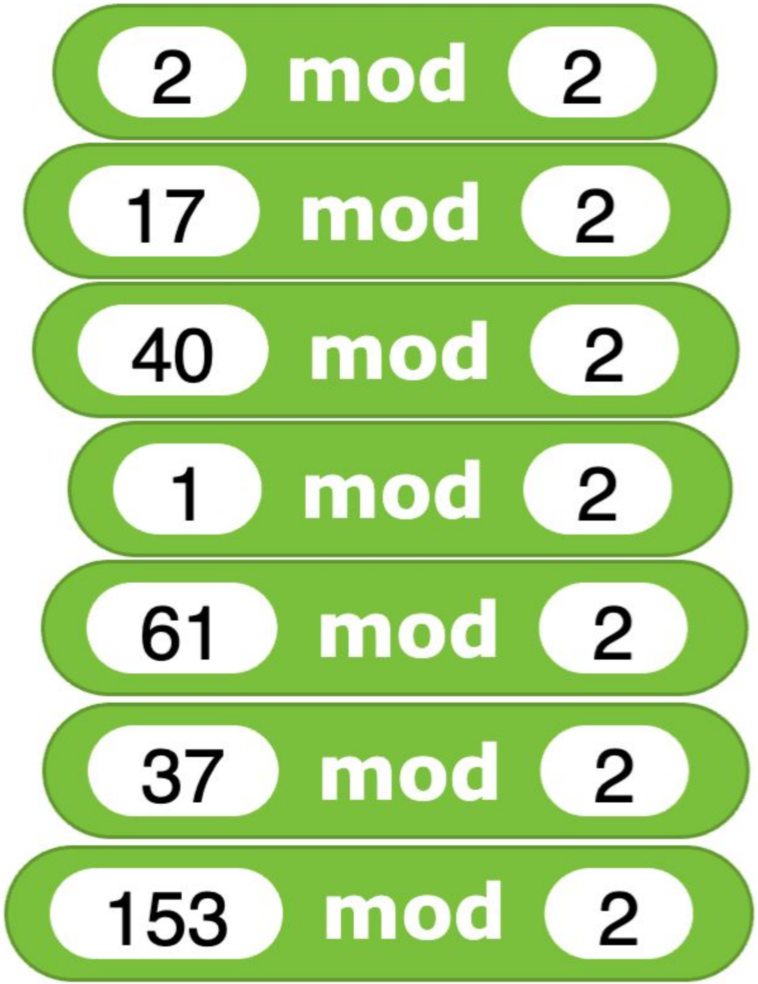
153 mod 2

= 0 (even)

= 1 (odd)

= 0 (even)

= 1 (odd)



2 mod 2 = 0 (even)

17 mod 2 = 1 (odd)

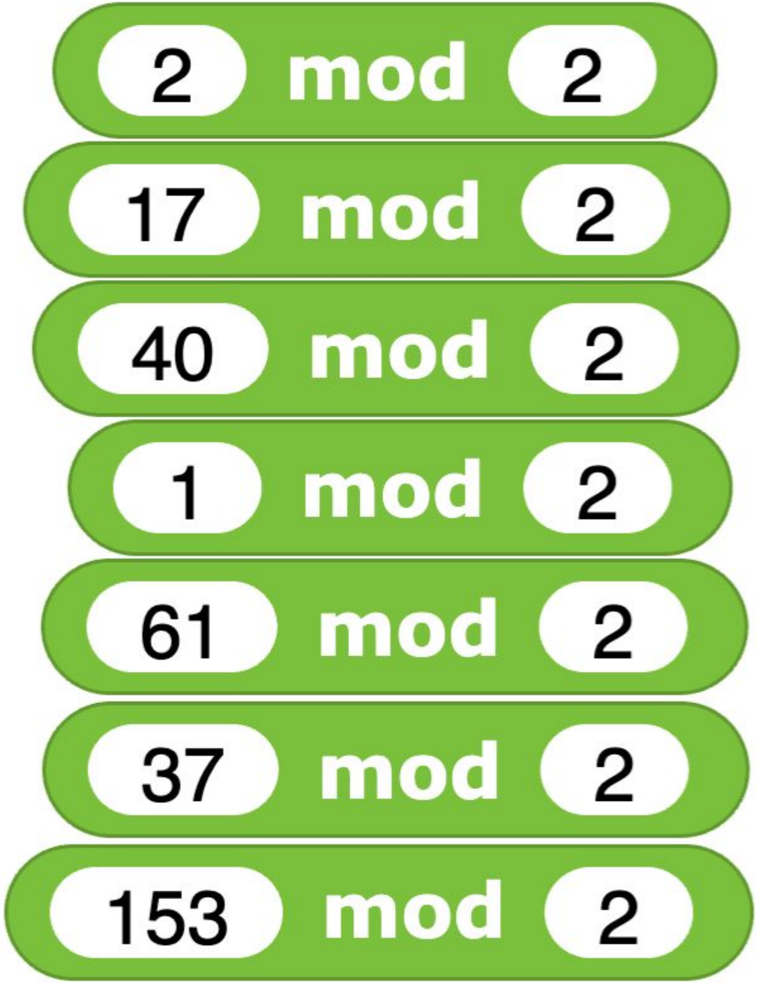
40 mod 2 = 0 (even)

1 mod 2 = 1 (odd)

61 mod 2 = 1 (odd)

37 mod 2

153 mod 2



2 mod 2 = 0 (even)

17 mod 2 = 1 (odd)

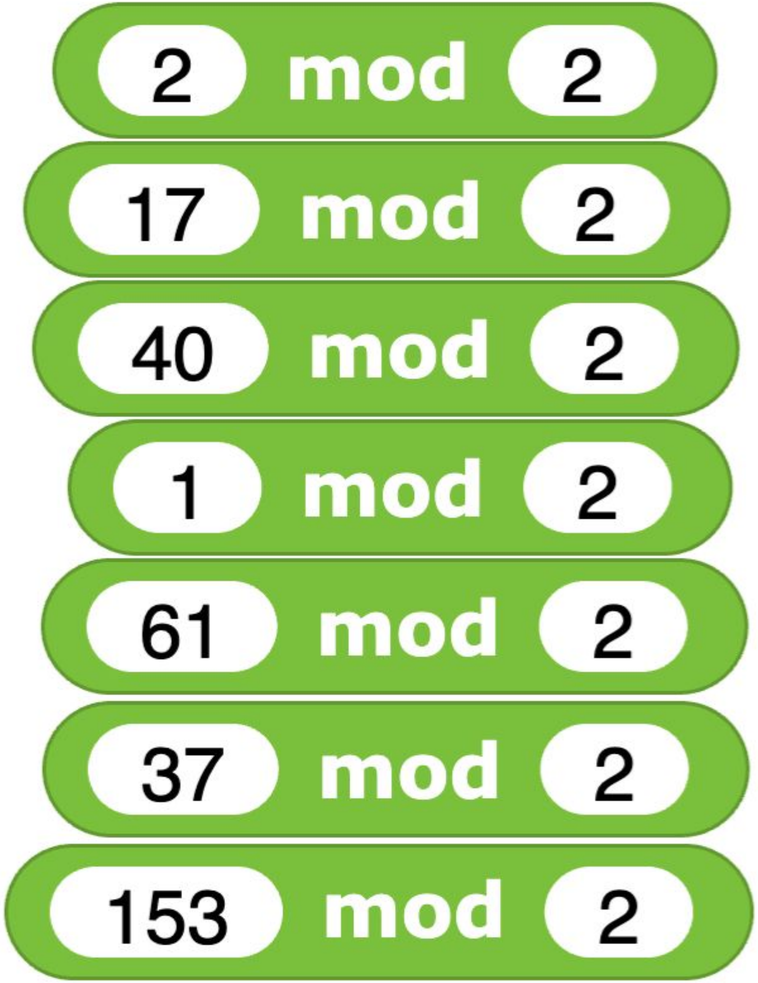
40 mod 2 = 0 (even)

1 mod 2 = 1 (odd)

61 mod 2 = 1 (odd)

37 mod 2 = 1 (odd)

153 mod 2



2 mod 2 = 0 (even)

17 mod 2 = 1 (odd)

40 mod 2 = 0 (even)

1 mod 2 = 1 (odd)

61 mod 2 = 1 (odd)

37 mod 2 = 1 (odd)

153 mod 2 = 1 (odd)