

AP Review 2: Programming (Part 2)

AP CSP REFERENCE SHEET

[HERE](#) is the **AP CSP Reference Sheet** that you **will have access to on test day**, which shows all commands that might appear on the test in both text-based and block-based pseudocode.

Relational & Boolean Expressions

Relational and **Boolean** operators return *either true or false*. The **true** or **false** values that get returned are called **Boolean values**, or just **Booleans**.

Relational operators from the [Reference Sheet](#):

Text and Block:

`a = b`
`a ≠ b`
`a > b`
`a < b`
`a ≥ b`
`a ≤ b`

The relational operators `=`, `≠`, `>`, `<`, `≥`, and `≤` are used to test the relationship between two variables, expressions, or values. A comparison using relational operators evaluates to a Boolean value.

For example, `a = b` evaluates to **true** if `a` and `b` are equal; otherwise it evaluates to **false**.

Relational operator examples and their Boolean values:

Example	Value
<code>7 > 4</code>	<code>true</code>
<code>4 > 7</code>	<code>false</code>
<code>6 = 6</code>	<code>true</code>
<code>10 ≤ 5</code>	<code>false</code>
<code>5 ≤ 5</code>	<code>true</code>
<code>5 ≠ 6</code>	<code>true</code>
<code>5 ≠ 5</code>	<code>false</code>
<code>true = true</code>	<code>true</code>
<code>false = true</code>	<code>false</code>

Boolean operators from the [Reference Sheet](#):

Text: NOT condition Block: NOT (condition)	Evaluates to true if condition is false ; otherwise evaluates to false .
Text: condition1 AND condition2 Block: (condition1) AND (condition2)	Evaluates to true if both condition1 and condition2 are true ; otherwise evaluates to false .
Text: condition1 OR condition2 Block: (condition1) OR (condition2)	Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true ; otherwise evaluates to false .

Relational operator examples and their Boolean values:

Example	Value	Explanation
(7 > 4) AND (1 = 1)	true	The AND operator only returns true if both parts are true ; 7 > 4 is true and 1 = 1 is true, so the overall AND statement is true
(1 < 5) AND (5 ≥ 7)	false	The AND operator only returns true if both parts are true ; 1 < 5 is true but 5 ≥ 7 is false , so the overall AND statement is false
(4 ≤ 3) OR (5 ≠ 6)	true	The OR operator only returns true if either or both parts are true ; 4 ≤ 3 is false but 5 ≠ 6 is true, so the overall OR statement is true since only one part needs to be true
(10 > 6) OR (true)	true	The OR operator only returns true if either or both parts are true ; 10 > 6 is true and true is true (of course!), so the overall OR statement is true since both parts are true
(1 = 2) OR (false = true)	false	The OR operator only returns true if either or both parts are true ; 1 = 2 is false and false = true is also false, so the overall OR statement is false since both parts are false
NOT (3 > 4)	true	The NOT operator reverses the Boolean value of whatever is inside, and 3 > 4 is false, so NOT(false) is true
NOT (4 > 3)	false	The NOT operator reverses the Boolean value of whatever is inside, and 4 > 3 is true, so NOT(true) is false

Use the [Reference Sheet](#) as needed to help you answer the following questions!

This line of code has executed:

num ← 5

With **num** having the value of 5, determine the boolean value (true or false) of the following statements:

- a.** num = 5
- b.** 6 > num
- c.** num < 4
- d.** (2 x 10) ≥ (4 x num)
- e.** LENGTH([2, 4, 6]) ≠ num
- f.** (num < 10) AND (num > 2)
- g.** (num > 2) AND (num > 6)
- h.** (num < 5) AND (num > 3)
- i.** (num > 8) AND (num < 2)
- j.** (num < 10) OR (num > 2)
- k.** (num > 2) OR (num > 6)
- l.** (num < 5) OR (num > 3)
- m.** (num > 8) OR (num < 2)
- n.** NOT(num > 10)
- o.** NOT(num < 6)
- p.** NOT(true = false)

Values (true or false):

- a.
- b.
- c.
- d.
- e.
- f.
- g.
- h.
- i.
- j.
- k.
- l.
- m.
- n.
- o.
- p.

[Check!](#)

What will the following algorithm display?

```
x ← 20
y ← false
IF ((x > 15) AND (NOT(y)))
{
    IF ((x < 10) = y)
    {
        DISPLAY("A")
    }
    ELSE
    {
        DISPLAY("B")
    }
}
ELSE
{
    DISPLAY("C")
}
DISPLAY("D")
```

[Check](#)

What will the following algorithms display?

Algorithm A

```
x ← 5
IF ((x + 4) > 8)
{
    IF (x < 10)
    {
        DISPLAY("A")
    }
    DISPLAY("B")
}
ELSE
{
    IF ((x * x) < 30)
    {
        DISPLAY("C")
    }
    DISPLAY("D")
}
```

Algorithm B

```
x ← 5
IF ((x + 4) > 8)
{
    IF (x > 9)
    {
        DISPLAY("A")
    }
    DISPLAY("B")
}
IF ((x * x) < 30)
{
    DISPLAY("C")
}
DISPLAY("D")
```

Algorithm A displays:

Algorithm B displays:

[Check](#)

<p>Here's a code segment:</p> <pre> a ← 3 b ← 5 c ← (a < b) AND ((a = 2) OR (b > 4)) DISPLAY(c) </pre>	<p>What is displayed? (i.e. what is the value of c) (<i>hint: c is either true or false!</i>)</p>
--	--

[Check!](#)

<p>The following code segment executes:</p> <pre> age ← 15 gpa ← 92.5 isRaining ← true isAdult ← (age > 18) </pre> <p>Determine the value (true or false) of each expression below <i>after</i> the code above executes:</p>	
A. isRaining	
B. isAdult	
C. (age > 10) AND (GPA < 90)	
D. (age < 18) AND ((GPA + 5) > 95)	
E. (gpa < age) AND (isAdult)	
F. NOT(isAdult) AND (isRaining)	
G. (gpa = 95) OR (age = 15)	
H. ((age - 5) ≤ 10) OR (isAdult)	
I. (16 > age) OR (80 < gpa)	
J. NOT(isRaining) OR (isAdult)	
K. NOT(GPA < 92) AND NOT(age < 15)	
L. NOT((isRaining) OR (10 < age))	
<p>M. What will be displayed?</p> <pre> IF ((age > 16) AND (isAdult)) { DISPLAY("Can drive and an adult") } </pre>	

<pre> ELSE { IF ((age > 16) OR (isAdult)) { IF (age > 16) { DISPLAY("Can drive") } IF (isAdult) { DISPLAY("Adult") } } ELSE { DISPLAY("Can't drive and not and adult") } } </pre>	
---	--

[Check Set 1](#)

<p>The same code segment executes:</p> <pre> age ← 15 gpa ← 92.5 isRaining ← true isAdult ← (age > 18) </pre> <p>Determine the value (true or false) of each expression below <i>after</i> the code above executes:</p>	
N. (age > 10) AND ((isRaining) AND (gpa < 95))	
O. (isRaining) AND ((gpa > 95) AND (age < 16))	
P. (isAdult) OR ((age > 10) AND (gpa < 95))	
Q. ((isRaining) AND (age = 10)) OR (isRaining)	
R. (age = 10) OR ((age = 15) OR (age = 20))	
S. NOT(isAdult) AND (NOT(gpa < 90))	
T. ((18 > age) AND (isAdult)) OR ((age > 15) AND (isRaining))	
u. ((18 > age) OR (isAdult)) AND ((age > 15) OR (isRaining))	
V. ((18 > age) AND (isAdult)) OR ((age > 15) OR (isRaining))	
W. NOT(((18 > age) AND (isAdult)) AND ((age > 15) OR (isRaining)))	

[Check Set 2](#)

Practice AP Question

Brooklyn Technical High School has academic data on every student, including `overallAverage`, `numberOfCredits`, and `absences`. The following expression is used to determine eligibility for a scholarship to a college of choice:

`(overallAverage > 89) AND ((numberOfCredits > 34) OR (absences < 5))`

Which of the following **two** sets of values would make the student eligible for the scholarship?

Select exactly **two**.

- A. `overallAverage = 90`, `numberOfCredits = 34`, `absences = 5`
- B. `overallAverage = 90`, `numberOfCredits = 35`, `absences = 3`
- C. `overallAverage = 95`, `numberOfCredits = 32`, `absences = 1`
- D. `overallAverage = 85`, `numberOfCredits = 32`, `absences = 0`

My answer:
(select two!):

and

After checking answer,
the correct answers are:

and

[Check your answer!](#)

Practice AP Question

An office building has two floors. A computer program is used to control an elevator that travels between the two floors. Physical sensors are used to set the following Boolean variables.

Variable	Description
<code>onFloor1</code>	set to <code>true</code> if the elevator is stopped on floor 1; otherwise set to <code>false</code>
<code>onFloor2</code>	set to <code>true</code> if the elevator is stopped on floor 2; otherwise set to <code>false</code>
<code>callTo1</code>	set to <code>true</code> if the elevator is called to floor 1; otherwise set to <code>false</code>
<code>callTo2</code>	set to <code>true</code> if the elevator is called to floor 2; otherwise set to <code>false</code>

The elevator moves when the door is closed and the elevator is called to the floor that it is not currently on. Which of the following Boolean expressions can be used in a selection statement to cause the elevator to move?

- (A) `(onFloor1 AND callTo2) AND (onFloor2 AND callTo1)`
- (B) `(onFloor1 AND callTo2) OR (onFloor2 AND callTo1)`
- (C) `(onFloor1 OR callTo2) AND (onFloor2 OR callTo1)`
- (D) `(onFloor1 OR callTo2) OR (onFloor2 OR callTo1)`

My answer:

After checking answer,
the correct answer is:

[Check your answer!](#)

Practice AP Question

What will be displayed if the following code segment is executed?

```
x ← 4
y ← x
IF (x ≤ y)
{
  IF (x ≠ y)
  {
    DISPLAY ("A")
  }
  ELSE
  {
    DISPLAY ("B")
  }
  DISPLAY ("C")
}
ELSE
{
  DISPLAY ("D")
}
```

- A. "A" followed by "C"
- B. "B" followed by "C"
- C. "C" followed by "D"
- D. "D" only
- E. None of the above

My answer:

*After checking answer,
the correct answer is:*

[Check your answer!](#)

12. Practice AP Question

Which of the following Boolean expressions are equivalent to $\text{num} \leq 23$?

Select two answers.

- A. $\text{num} < 23$ OR $\text{num} = 23$
- B. NOT ($\text{num} > 22$)
- C. NOT ($\text{num} > 23$)
- D. NOT ($\text{num} < 23$ AND $\text{num} = 23$)

My answers:

and

After checking answer,
the correct answers are:

and

[Check your answer!](#)

Practice AP Question

A taxi cab is willing to drive at most 50 miles in a trip, and charges \$2 per mile, plus a surcharge of \$10 on trips between 20 miles and 40 miles, and a \$15 surcharge on trips over 40 miles.

Two procedures are defined below:

- The `taxiDrive` procedure takes a parameter, `distance`, and returns `true` if the taxi will drive that far and `false` if it will not.
- The `taxiCost` procedure also takes a parameter, `distance`, and returns the cost of the trip.

```
PROCEDURE taxiDrive(distance)
{
  IF (distance > 50)
  {
    RETURN false
  }
  ELSE
  {
    RETURN true
  }
}
```

```
PROCEDURE taxiCost(distance)
{
  IF (distance < 20)
  {
    RETURN (distance * 2)
  }
  ELSE
  {
    IF ((distance ≥ 20) AND (distance ≤ 40))
    {
      RETURN (distance * 2 + 10)
    }
    ELSE
    {
      RETURN (distance * 2 + 15)
    }
  }
}
```

Determine what is displayed when the code segment below is executed:

```

trip ← 35
totalCost ← 0
IF (taxiDrive(trip))
{
    totalCost ← taxiCost(trip)
}
DISPLAY("The cost of the trip is " + totalCost)

```

- A. The cost of the trip is 0.
- B. The cost of the trip is 70.
- C. The cost of the trip is 80.
- D. The cost of the trip is 85.

My answer:

After checking answer,
the correct answer is:

[Check your answer!](#)

Arithmetic Operators & MOD

Here are the arithmetic operators you need to know for the exam from the [Reference Sheet](#):

Text and Block:

$a + b$
 $a - b$
 $a * b$
 a / b

The arithmetic operators $+$, $-$, $*$, and $/$ are used to perform arithmetic on a and b .

For example, $17 / 5$ evaluates to 3.4 .

The order of operations used in mathematics applies when evaluating expressions.

Text and Block:

$a \text{ MOD } b$

Evaluates to the remainder when a is divided by b . Assume that a is an integer greater than or equal to 0 and b is an integer greater than 0 .

For example, $17 \text{ MOD } 5$ evaluates to 2 .

The MOD operator has the same precedence as the $*$ and $/$ operators.

The **MOD operator** returns the **remainder** when the *first* input is divided by the *second*. For example, $17 \text{ MOD } 5 = 2$ because when 17 is divided by 5, the *remainder* is 2 (5 goes into 17 three times, remainder 2, which is the MOD). When one number divides another *evenly*, the *remainder* is 0. So for example, $15 \text{ MOD } 3 = 0$ because $15 / 3 = 5$ with *no* remainder!

Quick MOD Practice

Determine the value of each expression (#1-16 will be a number, #17- 20 will be true/false)

#	Expression	Ans.	#	Expression	Ans.
1	10 MOD 1		11	10 MOD 15	
2	10 MOD 2		12	10 MOD 30	
3	10 MOD 3		13	18 MOD 5	
4	10 MOD 4		14	5 MOD 18	
5	10 MOD 5		15	(3 MOD 2) + (6 MOD 4)	
6	10 MOD 6		16	15 MOD (12 MOD 5))	
7	10 MOD 7		17	(10 MOD 2) = 0	
8	10 MOD 8		18	(14 MOD 4) = (13 MOD 3)	
9	10 MOD 9		19	(25 MOD 13) = (12 MOD 24)	
10	10 MOD 10		20	(12 MOD 2) > (11 MOD 2)	

[Check your answers!](#)

Determine what is displayed when this code segment below is executed (**reminder!** [] means an empty list):

```
finalList ← []
nums ← [4, 15, 10, 9, 20, 1, 17, 13]
FOR EACH num IN nums
{
  IF ((num MOD 3) = 1)
  {
    APPEND(finalList, num)
  }
}
DISPLAY(finalList)
```

What gets displayed should be a list!
Write it using [] notation:

[Check!](#)

When working with **mathematical operations**, the same **order of operations** (PEMDAS) from math apply!

What would the expression $(4 * 10) - 2$ evaluate to?

What would the expression $4 * (10 - 2)$ evaluate to?

What would the expression $12 + 8 / 4$ evaluate to?

How about $2 * 3 + 4 / 2 - 1$?

What would $30 - 14 \text{ MOD } 5 * 7 \text{ MOD } 4$ evaluate to?
(MOD has the *same* precedence as * and /, i.e. MOD, *, / left to right)

[Check my math](#)

AP EXAM PRACTICE QUESTION #1

The following incomplete code fragment was designed to test if number is odd:

```
IF (<MISSING CONDITION>)  
{  
    DISPLAY ("The number is odd!")  
}
```

Which of the following can be used in place of <MISSING CONDITION> ?

X the correct answer choice.

<input type="checkbox"/>	A. <code>number MOD 1 = 0</code>
<input type="checkbox"/>	B. <code>number MOD 1 = 1</code>
<input type="checkbox"/>	C. <code>number MOD 2 = 0</code>
<input type="checkbox"/>	D. <code>number MOD 2 = 1</code>

[Check my answer](#)

AP EXAM PRACTICE QUESTION

Consider the following code segment:

```
i ← 100
repeat until i = 0
{
    <MISSING CODE>
}
```

Which of the following replacements for <MISSING CODE> will cause an infinite loop?

X the correct answer choice.

<input type="checkbox"/>	A. $i \leftarrow i \bmod 2$
<input type="checkbox"/>	B. $i \leftarrow i \bmod 3$
<input type="checkbox"/>	C. $i \leftarrow i \bmod 4$
<input checked="" type="checkbox"/>	D. $i \leftarrow i \bmod 5$

[Check my answer](#)

AP EXAM PRACTICE QUESTION

A local coffee shop started passing out punch cards that, for each time you purchase a coffee, they punch a hole in the card. When you get six punched holes, you get a free coffee! A software programmer wanted to write a program to display a message to the customer on the cash register when they have earned their free coffee. Which of the following segments of code could be used to display this message at the right time?

X the correct answer choice.

A.

```
IF (punchedHoles / 6) = 0
{
    DISPLAY("Free coffee!")
}
```

B.

```
IF NOT((punchedHoles / 6) = 0)
{
    DISPLAY("Free coffee!")
}
```

C.

```
IF (punchedHoles MOD 6) = 0
{
    DISPLAY("Free coffee!")
}
```

D.

```
IF NOT((punchedHoles MOD 6) = 0)
{
    DISPLAY("Free coffee!")
}
```

[Check my answer](#)

AP EXAM VOCABULARY

A **software library** is a collection of functions that can be used in programs.

Using **libraries** can simplify the development of new programs. When you use functions that you already know work correctly, you reduce the amount of time you need to spend coding, the number of possible bugs your code can have, and how much of your project you need to test.

Python has a bunch of **libraries** that you could import and use in your projects, and in Python we often call them "**modules**" that get imported using the import statement.

Some examples of libraries/modules that we have used this year are `math`, `turtle`, and `os`:

```
import math
import turtle
import os
```

An **application programming interface (API)** is a fancy term used in the world of computer science to mean the **documentation** about what the functions in a software library are used for and describes the inputs and outputs to functions. This **documentation** can help someone *else* (i.e. another programmer) use the functions in your library -- *without having to look at the code to try and understand what it does*.

When you add comments above a function that explains what the function does, you are providing the "**API**" for that function:

```
# draws a row of 6 bricks for type "A" and a row
# of 5 bricks with two partial bricks for type "B"
def draw_row(row_type):
    if row_type == "A":
        for i in range(5):
            draw_brick(30, 10)
            t.penup()
            t.forward(40)
            t.pendown()
```

This **documentation** tells someone else reading your code what the function does; this is the "**API**" for the function!

Robot Programming

(a.k.a. “Robots in a maze” problems)

Here are the **Robot** commands from the [Reference Sheet](#):

If the robot attempts to move to a square that is not open or is beyond the edge of the grid, the robot will stay in its current location and the program will terminate.

Text:
MOVE_FORWARD ()
Block:

MOVE_FORWARD

The robot moves one square forward in the direction it is facing.

Text:
ROTATE_LEFT ()
Block:

ROTATE_LEFT

The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).

Text:
ROTATE_RIGHT()
Block:

ROTATE RIGHT

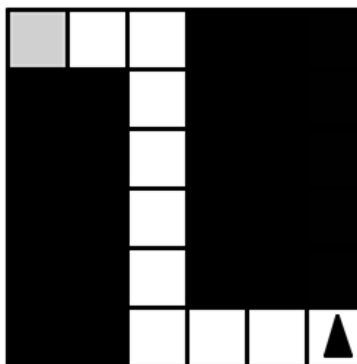
The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).

Text:
CAN_MOVE(direction)
Block:
CAN_MOVE direction

Evaluates to **true** if there is an open square one square in the direction relative to where the robot is facing; otherwise evaluates to **false**. The value of **direction** can be **left**, **right**, **forward**, or **backward**.

AP EXAM PRACTICE QUESTION

The figure below shows a robot in a grid of squares. The robot is represented as a triangle, which is initially facing upward. The robot can move into a white or gray square but cannot move into a black region.



Consider the procedure TurnAndMove below that has two parameters, numberOfTurns and numberOfMoves:


```

PROCEDURE TurnAndMove (numberOfTurns, numberOfMoves)
{
    REPEAT numberOfTurns TIMES
    {
        ROTATE_LEFT ()
    }
    REPEAT numberOfMoves TIMES
    {
        MOVE_FORWARD ()
    }
}

```

Which of the following code segments can be used to move the robot to the gray square?

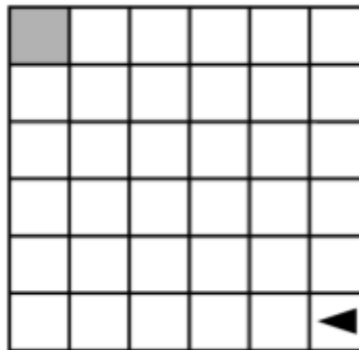
X the correct answer choice:

	(A) TurnAndMove (1, 3) TurnAndMove (1, 5) TurnAndMove (1, 2)
	(B) TurnAndMove (1, 3) TurnAndMove (3, 5) TurnAndMove (1, 2)
	(C) TurnAndMove (1, 4) TurnAndMove (3, 6) TurnAndMove (1, 3)
	(D) TurnAndMove (3, 1) TurnAndMove (5, 3) TurnAndMove (2, 1)

[Check answer](#)

AP EXAM PRACTICE QUESTION

The figure below shows a robot in a grid of squares. The robot is represented as a triangle, which is initially facing left and in the bottom right corner.



Which of the following code segments can be used to move the robot to the gray square?

Select **two** answers.

X the TWO correct answer choices:

(A) REPEAT 5 TIMES
{
 MOVE_FORWARD ()
 ROTATE_RIGHT ()
}
REPEAT 5 TIMES
{
 MOVE_FORWARD ()
 ROTATE_LEFT ()
}

(B) REPEAT 5 TIMES
{
 MOVE_FORWARD ()
 ROTATE_RIGHT ()
 MOVE_FORWARD ()
 ROTATE_LEFT ()
}

(C) REPEAT 5 TIMES
{
 REPEAT 2 TIMES
 {
 MOVE_FORWARD ()
 }
 ROTATE_RIGHT ()
}

```

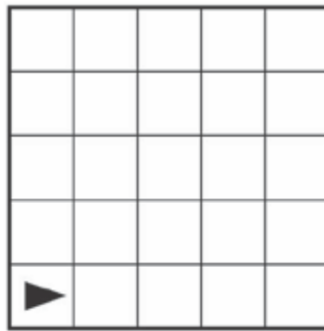
(D) REPEAT 2 TIMES
{
    REPEAT 5 TIMES
    {
        MOVE_FORWARD ( )
    }
    ROTATE_RIGHT ( )
}

```

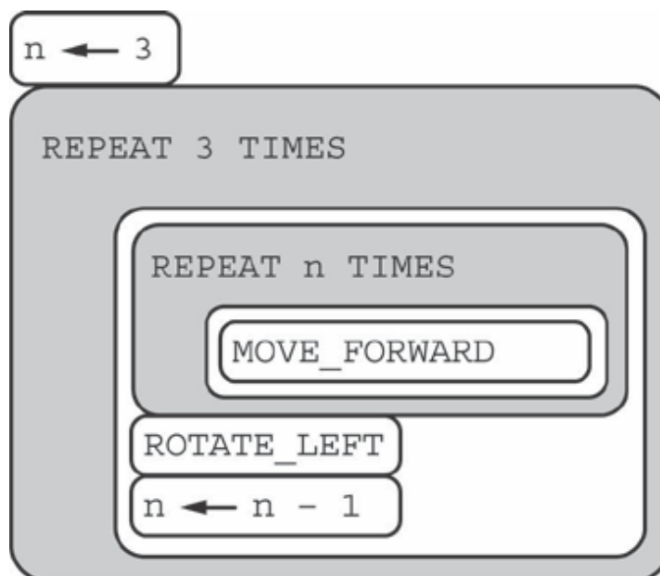
[Check answer](#)

AP EXAM PRACTICE QUESTION

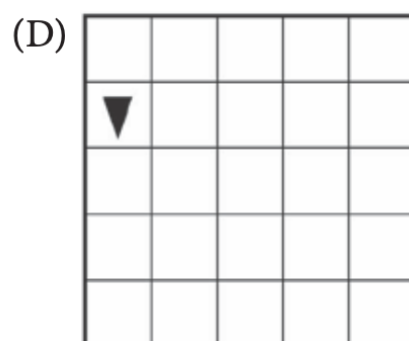
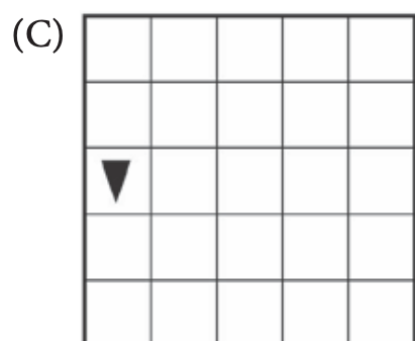
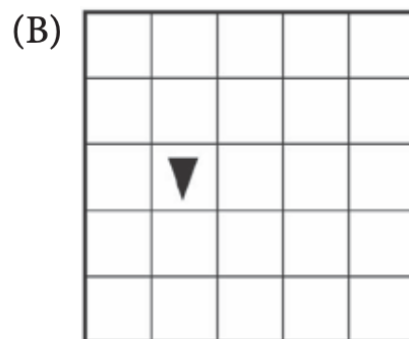
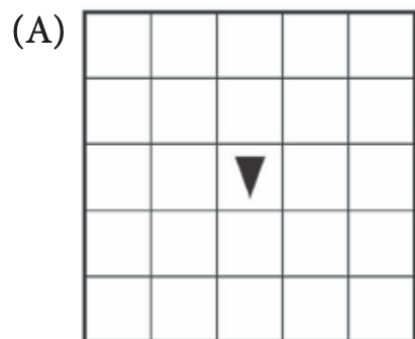
The following question uses a robot in a grid of squares. The robot is represented as a triangle, which is initially in the bottom left square of the grid and facing right.



Consider the following code segment, which moves the robot in the grid.



Which of the following shows the location of the robot after running the code segment?



AP Exam Pro Tip (you will want to do this!): Sketch it out on scrap paper 😎

X the correct answer choice.

<input type="checkbox"/>	(A)
<input type="checkbox"/>	(B)
<input type="checkbox"/>	(C)
<input type="checkbox"/>	(D)

[Check answer](#)

Let's review this **Robot** command:

Text: <code>CAN_MOVE(direction)</code> Block: <code>CAN_MOVE</code> <code>direction</code>	Evaluates to <code>true</code> if there is an open square one square in the direction relative to where the robot is facing; otherwise evaluates to <code>false</code> . The value of <code>direction</code> can be <code>left</code> , <code>right</code> , <code>forward</code> , or <code>backward</code> .
---	--

CAN_MOVE(right) : returns `true` if the block immediately to the Robot's **right** (when it's facing forward) is a white space; otherwise, it returns `false`.

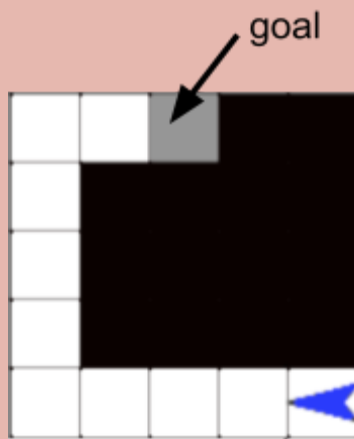
CAN_MOVE(left) : returns `true` if the block immediately to the Robot's **left** (when it's facing forward) is a white space; otherwise, it returns `false`.

CAN_MOVE(forward) : returns `true` if the block immediately in **front** of the Robot's (when it's facing forward) is a white space; otherwise, it returns `false`.

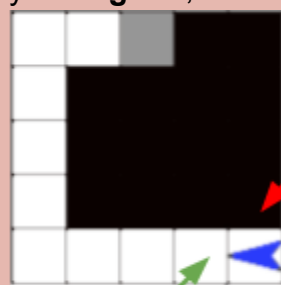
CAN_MOVE(backward) : returns `true` if the block immediately **behind** the Robot's (when it's facing forward) is a white space; otherwise, it returns `false`.

EXAMPLE 1

In the maze below, the Robot can move in **white** spaces, and the **gray** box is the **GOAL**:



The Robot is currently **facing left**, and at its **current** location, each of the predicates would report:



`CAN_MOVE(right)` = **false**

`CAN_MOVE(backward)` = **false**

`CAN_MOVE(forward)` = **true**

`CAN_MOVE(left)` = **false**

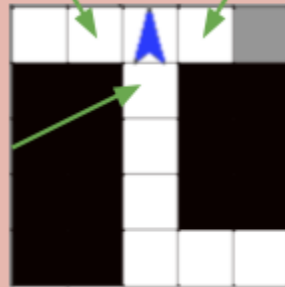
EXAMPLE 2: Here is a different situation:

`CAN_MOVE(forward) = false`

`CAN_MOVE(left) = true`

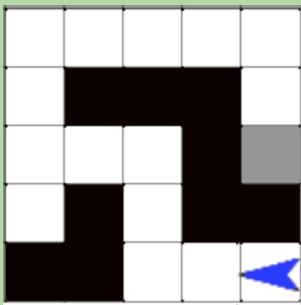
`CAN_MOVE(right) = true`

`CAN_MOVE(backward) = true`



For each maze situation below, determine what each of the four procedures would return: either TRUE or FALSE. The first one is done for you as an example

A.



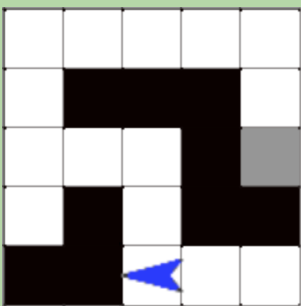
`CAN_MOVE(right) = false`

`CAN_MOVE(left) = false`

`CAN_MOVE(forward) = true`

`CAN_MOVE(backward) = false`

B.



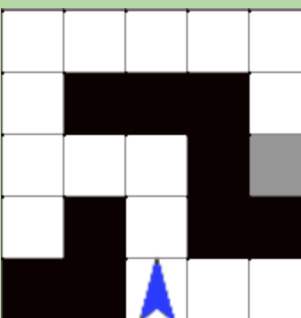
`CAN_MOVE(right) =`

`CAN_MOVE(left) =`

`CAN_MOVE(forward) =`

`CAN_MOVE(backward) =`

C.



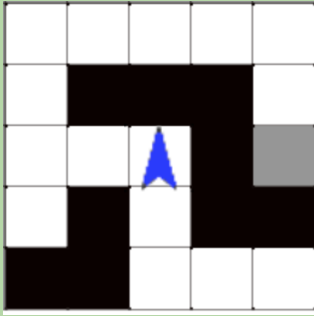
`CAN_MOVE(right) =`

`CAN_MOVE(left) =`

`CAN_MOVE(forward) =`

`CAN_MOVE(backward) =`

D.



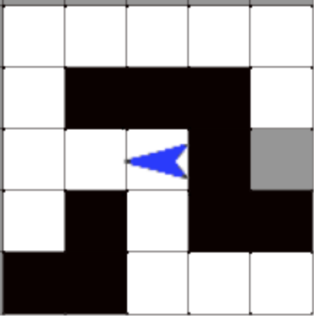
CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

E.



CAN_MOVE(right) =

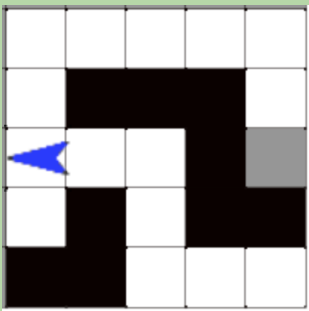
CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

CHECK SO FAR!

F.



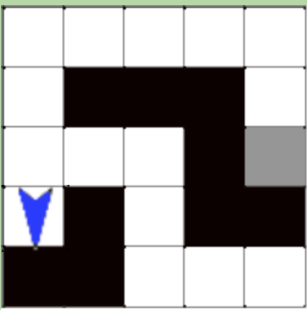
CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

G.



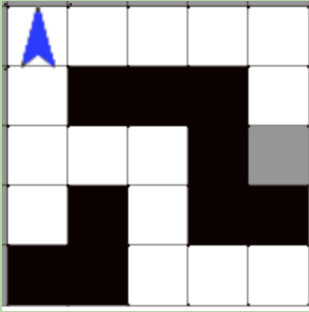
CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

H.



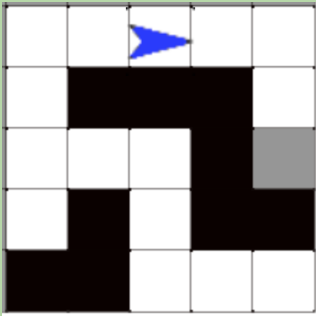
CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

I.



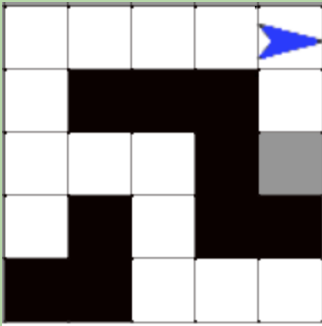
CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

J.



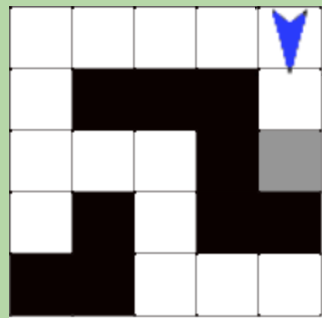
CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =

K.

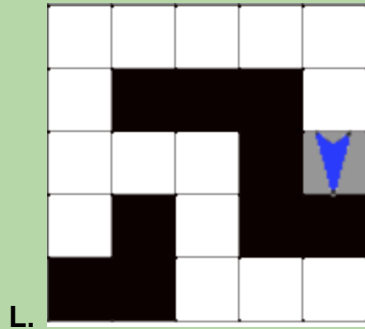


CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

CAN_MOVE(backward) =



L.

CAN_MOVE(right) =

CAN_MOVE(left) =

CAN_MOVE(forward) =

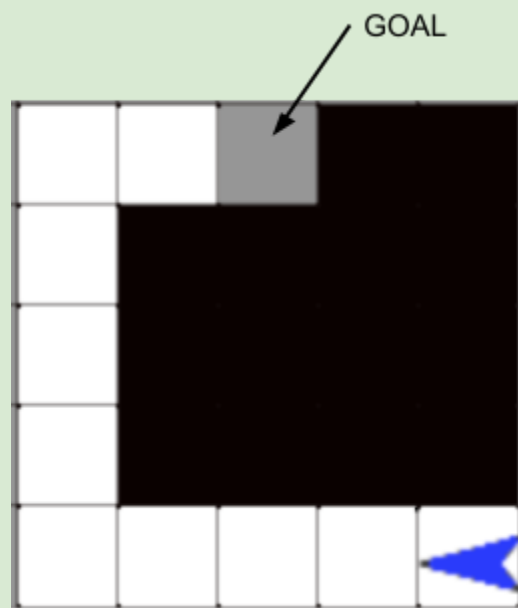
CAN_MOVE(backward) =

[CHECK AGAIN](#)

Analyze this code segment:

Would this code move the robot to the Goal?

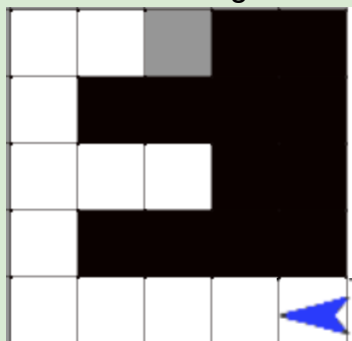
```
REPEAT UNTIL (goal_reached)
{
  IF (CAN_MOVE(forward))
  {
    MOVE_FORWARD()
  }
  IF (CAN_MOVE(left))
  {
    ROTATE_LEFT()
  }
  IF (CAN_MOVE(right))
  {
    ROTATE_RIGHT()
  }
}
```



Would the code above move the Robot in the maze above from its starting location to the Goal?

[Check your answer](#)

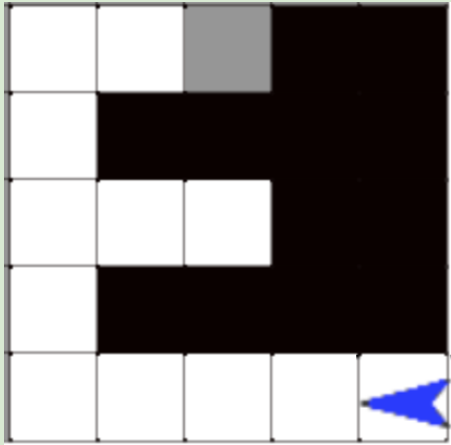
Would the same script above **also** work for getting the Robot in the maze below to its goal?



[Check your answer](#)

If the following code segment was executed, in which the `CAN_MOVE(left)` and `CAN_MOVE(right)` conditions have been **swapped** from the code in the previous problem, would the robot reach the goal (gray square)?

```
REPEAT UNTIL (goal_reached)
{
  IF (CAN_MOVE(forward))
  {
    MOVE_FORWARD()
  }
  IF (CAN_MOVE(right))
  {
    ROTATE_RIGHT()
  }
  IF (CAN_MOVE(left))
  {
    ROTATE_LEFT()
  }
}
```



Yes or no?

[Check my answer!](#)

AP PRACTICE QUESTION

Take a look at the two code segments (slightly different!) and the maze below:

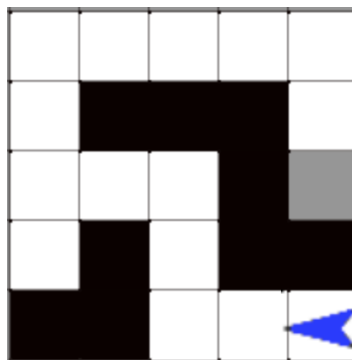
Code Segment A

```
REPEAT UNTIL (GoalReached)
{
  IF (CAN_MOVE(right))
  {
    ROTATE_RIGHT()
  }
  IF (CAN_MOVE(left))
  {
    ROTATE_LEFT()
  }
  IF (CAN_MOVE(forward))
  {
    MOVE_FORWARD()
  }
}
```

Code Segment B

```
REPEAT UNTIL (GoalReached)
{
  IF (CAN_MOVE(left))
  {
    ROTATE_LEFT()
  }
  IF (CAN_MOVE(right))
  {
    ROTATE_RIGHT()
  }
  IF (CAN_MOVE(forward))
  {
    MOVE_FORWARD()
  }
}
```

Maze :



Which of the two code segments would allow the Robot in the maze to reach the Goal?

- A. Code segment A, only
- B. Code segment B, only
- C. Both code segments A and B
- D. Neither code segment A nor B

Answer:

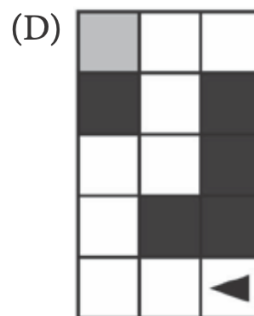
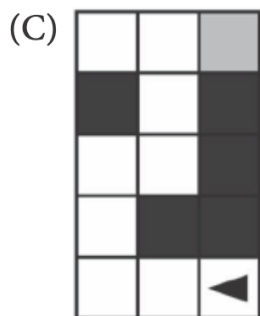
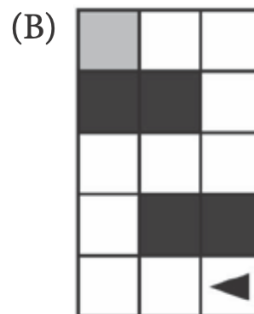
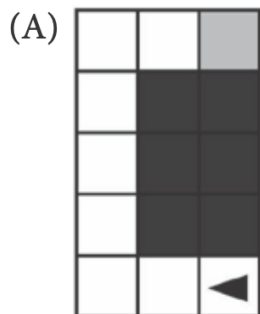
[Check the answer!](#)

AP EXAM PRACTICE QUESTION

The code segment below is intended to move a robot in a grid to a gray square. The program segment uses the procedure `GoalReached`, which evaluates to `true` if the robot is in the gray square and evaluates to `false` otherwise. The robot in each grid is represented as a triangle and is initially facing left. The robot can move into a white or gray square but cannot move into a black region.

```
REPEAT UNTIL (GoalReached ())
{
  IF (CAN_MOVE (forward))
  {
    MOVE_FORWARD ()
  }
  IF (CAN_MOVE (right))
  {
    ROTATE_RIGHT ()
  }
  IF (CAN_MOVE (left))
  {
    ROTATE_LEFT ()
  }
}
```

For which of the following grids does the code segment NOT correctly move the robot to the gray square?



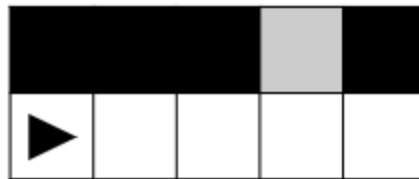
X the correct answer choice:

	(A)
	(B)
	(C)
	(D)

[Check answer](#)

AP EXAM PRACTICE QUESTION

The grid below contains a robot represented as a triangle, initially facing up. The robot can move into a white or gray square but cannot move into a black region.



Which two of the following code segments can be used to move the robot to the gray square?

Select two answers.

X the TWO correct answer choices.

	(A) REPEAT 3 TIMES { MOVE_FORWARD () } ROTATE_LEFT () MOVE_FORWARD ()
	(B) REPEAT 4 TIMES { MOVE_FORWARD () } ROTATE_LEFT () MOVE_FORWARD ()
	(C) IF (CAN_MOVE (left)) { ROTATE_LEFT () } MOVE_FORWARD ()

```

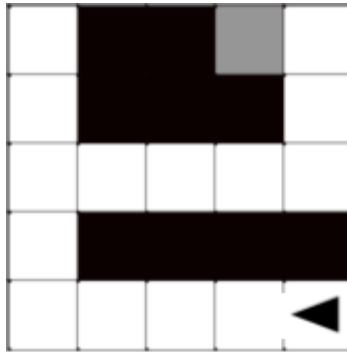
(D) REPEAT UNTIL (CAN_MOVE (left))
{
    MOVE_FORWARD ()
}
ROTATE_LEFT ()
MOVE_FORWARD ()

```

[Check answer](#)

AP EXAM PRACTICE QUESTION

The grid below contains a robot represented as a triangle, initially facing up. The robot can move into a white or gray square but cannot move into a black region.



The code segment below uses the procedure GoalReached, which evaluates to true if the robot is in the gray square and evaluates to false otherwise.

```

REPEAT UNTIL (GoalReached ())
{
    <MISSING CODE>
}

```

Which of the following replacements for <MISSING CODE> can be used to move the robot to the gray square?

I.

```

IF (CAN_MOVE (forward))
{
    MOVE_FORWARD ()
}
IF (CAN_MOVE (right))
{
    ROTATE_RIGHT ()
}
IF (CAN_MOVE (left))
{
    ROTATE_LEFT ()
}

```

II.

```

IF (CAN_MOVE (forward))
{
    MOVE_FORWARD ()
}
IF (CAN_MOVE (left))
{
    ROTATE_LEFT ()
}
IF (CAN_MOVE (right))
{
    ROTATE_RIGHT ()
}

```

X the correct answer choice:

<input type="checkbox"/>	(A) I, only
<input type="checkbox"/>	(B) II, only
<input type="checkbox"/>	(C) Both I and II
<input type="checkbox"/>	(D) Neither I nor II

[Check answer](#)

DONE WITH PACKET 2

Determine the value (true or false) of the following statements.

Assume this code runs: `num ← 5`

- a. `num = 5`
- b. `6 > num`
- c. `num < 4`
- d. `(2 x 10) ≥ (4 x num)`
- e. `LENGTH([2, 4, 6]) ≠ num`
- f. `(num < 10) AND (num > 2)`
`true` `true`
- g. `(num > 2) AND (num > 6)`
`true` `false`
- h. `(num < 5) AND (num > 3)`
`false` `true`
- i. `(num > 8) AND (num < 2)`
`false` `false`
- j. `(num < 10) OR (num > 2)`
`true` `true`
- k. `(num > 2) OR (num > 6)`
`true` `false`
- l. `(num < 5) OR (num > 3)`
`false` `true`
- m. `(num > 8) OR (num < 2)`
`false` `false`
- n. `NOT(num > 10)`
`false`
- o. `NOT(num < 6)`
`true`
- p. `NOT(true = false)`
`false`

- a. `true` (num is 5, and 5 does equal 5)
- b. `true` (6 is greater than 5)
- c. `false` (num is not less than 4)
- d. `true` (20 is greater than or equal to 20)
- e. `true` (the length of the list is 3 which does not equal num, which is 5)
- f. `true` (both parts of AND are true)
- g. `false` (one part of AND is false)
- h. `false` (one part of AND is false)
- i. `false` (both parts of AND are false)
- j. `true` (both parts of OR are true)
- k. `true` (at least one part of OR is true)
- l. `true` (at least one part of OR is true)
- m. `false` (both parts of OR are false)
- n. `true` (the part in the NOT is false)
- o. `false` (the part in the NOT is true)
- p. `true` (the part in the NOT is false)

Answer ([back](#))

What will the following algorithm display?

```
x ← 20
y ← false
IF ((x > 15) AND (NOT(y)))
{
  IF ((x < 10) = y)
  {
    DISPLAY("A")
  }
  ELSE
  {
    DISPLAY("B")
  }
}
ELSE
{
  DISPLAY("C")
}
DISPLAY("D")
```

It displays **two** letters: **A then D**

The first if statement condition is TRUE since it's an AND statement and *both* parts are true:

$$\begin{array}{rcl} & \text{NOT}(y) & = \text{NOT}(\text{false}) \\ \text{true} & & = \text{true} \\ ((\mathbf{x} > \mathbf{15}) \text{ AND } (\mathbf{NOT}(\mathbf{y}))) & & \end{array}$$

Since the overall

$((\mathbf{x} > \mathbf{15}) \text{ AND } (\mathbf{NOT}(\mathbf{y})))$
condition is true, the code goes *into* the first if statement and comes to the second if statement condition, which is ALSO true!

$$\begin{array}{rcl} \text{false} & \text{false} & \\ ((\mathbf{x} < \mathbf{10}) = \mathbf{y}) & & \end{array}$$

This is **TRUE** because $\text{false} = \text{false}$ is *true*

Therefore, it reaches `DISPLAY("A")` and displays **A**

Then, it skips `DISPLAY("B")` and `DISPLAY("C")` since those are in the else statements.

It reaches `DISPLAY("D")` which is *after* the if-else statement, so it displays **D**.

Answer ([back](#))

What will the following algorithms display?

Algorithm A

```
x ← 5
IF ((x + 4) > 8) true
{
  IF (x < 10) true
  {
    DISPLAY("A")
  }
  DISPLAY("B")
}
ELSE (this code doesn't run
since it's part of the else)
{
  IF ((x * x) < 30)
  {
    DISPLAY("C")
  }
  DISPLAY("D")
}
```

Algorithm B

```
x ← 5
IF ((x + 4) > 8) true
{
  IF (x > 9) false
  {
    DISPLAY("A")
  }
  DISPLAY("B")
}
IF ((x * x) < 30) true
{
  DISPLAY("C")
}
DISPLAY("D")
```

Algorithm A displays:

A
B

Note that **DISPLAY("B")** is under the if statement, so it runs). Neither C nor D display since they are in the *else* part (which doesn't execute since we executed the *if* part).

Algorithm B displays:

B
C
D

Note there are *two separate if statements* -- **NOT** an if-else statement! The code inside both if statements execute since both conditions are true.

Answer ([back](#))

Here's a code segment:

You evaluate INNER parentheses first, just like in math class!

`a ← 3`

`b ← 5`

`c ← (a < b) AND ((a = 2) OR (b > 4))`
 `(true) AND (false OR true)`
 `true AND true`
 `true`

(so c is assigned the value true)

`DISPLAY(c)`

What is displayed? (i.e. what is the value of `c`) (*hint: c is either true or false!*)

true

Answer ([back](#))

The following code segment executes:

```
age ← 15
gpa ← 92.5
isRaining ← true
isAdult ← (age > 18)
```

Determine the value (true or false) of each expression below *after* the code above executes:

A. isRaining	true
B. isAdult age > 18 is false, so this is false	false
C. (age > 10) AND (GPA < 90) = true AND false = false the overall AND is false since one part is false	false
D. (age < 18) AND ((GPA + 5) > 95) = true AND true = true the overall AND is true since both parts are true	true
E. (gpa < age) AND (isAdult) = false AND false = false the overall AND is false since both parts are false	false
F. NOT(isAdult) AND (isRaining) = NOT(false) AND true = true AND true = true the overall AND is true since both parts are true	true
G. (gpa = 95) OR (age = 15) = false OR true = true the overall OR is true since at least one part is true	true
H. ((age - 5) ≤ 10) OR (isAdult) = true OR false = true the overall OR is true since at least one part is true	true

I. (16 > age) OR (80 < gpa) = true OR true = true the overall OR is true since at least one part is true	true
J. NOT(isRaining) OR (isAdult) = NOT(true) OR false = false OR false = false the overall OR is false since both parts are false	false
K. NOT(GPA < 92) AND NOT(age < 15) = NOT(false) AND NOT(false) = true AND true = true the overall AND is true since both parts are true	true
L. NOT((isRaining) OR (10 < age)) = NOT(true OR true) = NOT(true) = false	false
M. What will be displayed? IF ((age > 16) AND (isAdult)) false { DISPLAY("Can drive and an adult") } ELSE (this part runs) { IF ((age > 16) OR (isAdult)) false { IF (age > 16) { DISPLAY("Can drive") } IF (isAdult) { DISPLAY("Adult") } } ELSE { DISPLAY("Can't drive and not and adult") } }	Can't drive and not and adult

[\(back\)](#)

Answers ([back](#))

N. (age > 10) AND ((isRaining) AND (gpa < 95)) = true AND (true AND true) = true AND true = true	true
O. (isRaining) AND ((gpa > 95) AND (age < 16)) = true AND (false AND true) = true AND false = false	false
P. (isAdult) OR ((age > 10) AND (gpa < 95)) = false OR (true AND true) = false OR true = true	true
Q. ((isRaining) AND (age = 10)) OR (isRaining) = true AND (false OR true) = true AND true = true	true
R. (age = 10) OR ((age = 15) OR (age = 20)) = false OR (true OR false) = false OR true = true	true
S. NOT(isAdult) AND (NOT(gpa < 90)) = NOT(false) AND NOT(false) = true AND true = true	true
T. ((18 > age) AND (isAdult)) OR ((age > 15) AND (isRaining)) = (true AND false) OR (false AND true) = false OR false = false	false
U. ((18 > age) OR (isAdult)) AND ((age > 15) OR (isRaining)) = (true OR false) AND (false OR true) = true AND true = true	true

V. ((18 > age) AND (isAdult)) OR ((age > 15) OR (isRaining)) = (true AND false) OR (false OR true) = false OR true = true	true
W. NOT(((18 > age) AND (isAdult)) AND ((age > 15) OR (isRaining))) = NOT((true AND false) AND (false OR true)) = NOT(false AND true) = NOT(false) = true	true

([back](#))

Answer ([back](#))

Correct Answers (select two): B & C

Practice AP Question

Brooklyn Technical High School has academic data on every student, including `overallAverage`, `numberOfCredits`, and `absences`. The following expression is used to determine eligibility for a scholarship to a college of choice:

$$(\text{overallAverage} > 89) \text{ AND } ((\text{numberOfCredits} > 34) \text{ OR } (\text{absences} < 5))$$

Which of the following **two** sets of values would make the student eligible for the scholarship?
Select exactly two.

- A. `overallAverage = 90`, `numberOfCredits = 34`, `absences = 5`
- B. `overallAverage = 90`, `numberOfCredits = 35`, `absences = 3`
- C. `overallAverage = 95`, `numberOfCredits = 32`, `absences = 1`
- D. `overallAverage = 85`, `numberOfCredits = 32`, `absences = 0`

Correct answers: B and C

For A: The OR statement is false, since 34 is not > 34 and 5 is not less than 5, so the overall AND statement is false.

For B: The OR statement is true, since $35 > 34$ and $3 < 5$ (although only one needs to be true), and overall average is 90, which is > 89 , so the overall AND statement is true.

For C: The OR statement is true, $1 < 5$ (it doesn't matter that number of credits was only 32 since only one needs to be true for an OR statement). Since the overall average is 95, which is > 89 , the overall AND statement is true.

For D: Although the OR statement is true, since $0 < 5$, the overall average is not > 89 so the overall AND statement is false.

Answer ([back](#))

Correct Answer: B

Practice AP Question

An office building has two floors. A computer program is used to control an elevator that travels between the two floors. Physical sensors are used to set the following Boolean variables.

Variable	Description
onFloor1	set to true if the elevator is stopped on floor 1; otherwise set to false
onFloor2	set to true if the elevator is stopped on floor 2; otherwise set to false
callTo1	set to true if the elevator is called to floor 1; otherwise set to false
callTo2	set to true if the elevator is called to floor 2; otherwise set to false

The elevator moves when the door is closed and the elevator is called to the floor that it is not currently on. Which of the following Boolean expressions can be used in a selection statement to cause the elevator to move?

(A) `(onFloor1 AND callTo2) AND (onFloor2 AND callTo1)`

(B) `(onFloor1 AND callTo2) OR (onFloor2 AND callTo1)`

(C) `(onFloor1 OR callTo2) AND (onFloor2 OR callTo1)`

(D) `(onFloor1 OR callTo2) OR (onFloor2 OR callTo1)`

The key to this problem is that **the elevator moves when the door is closed and the elevator is called to a floor that is different than it's currently on.**

So this happens when the elevator is either *on floor 1 and it gets called to floor 2*, **or** *it's on floor 2 and it gets called to floor 1*.

So the Boolean expression that represents when the elevator moves is:

(B) `(onFloor1 AND callTo2) OR (onFloor2 AND callTo1)`

Answer ([back](#))

Correct Answer: B

Practice AP Question

What will be displayed if the following code segment is executed?

```
x ← 4
y ← x
IF (x ≤ y)
{
    IF (x ≠ y)
    {
        DISPLAY ("A")
    }
    ELSE
    {
        DISPLAY ("B")
    }
    DISPLAY ("C")
}
ELSE
{
    DISPLAY ("D")
}
```

A. "A" followed by "C"

B. "B" followed by "C"

C. "C" followed by "D"

D. "D" only

E. None of the above

The correct answer is B.

Explanation

x is 4 and y is 4 (since it's assigned the same value as x)

$x \leq y$ is **true**, so the first part of the IF statement runs.

$x \neq y$ is **false**, because x and y are both 4, so $4 \neq 4$ is **false**! (note that its opposite, $4 = 4$, is **true**)

Since $x \neq y$ is false, the "nested" else executes, displaying **"B"**.

Then, **"C"** gets displayed since it's below the nested if-else statement.

"D" does *not* get displayed, since it's in the else part (and that doesn't execute since the if part did)

Answer ([back](#))

Correct answers: A & C

Practice AP Question

Which of the following Boolean expressions are equivalent to $\text{num} \leq 23$?

Select two answers.

A. $\text{num} < 23$ OR $\text{num} = 23$

B. NOT ($\text{num} > 22$)

C. NOT ($\text{num} > 23$)

D. NOT ($\text{num} < 23$ AND $\text{num} = 23$)

Explanations:

A. $\text{num} \leq 23$ is equivalent to $\text{num} < 23$ OR $\text{num} = 23$ because \leq is
"less than OR equal to!" $\rightarrow \text{num} < 23$ OR $\text{num} = 23$

C. $\text{num} \leq 23$ is equivalent to NOT ($\text{num} > 23$) because numbers that are less than or equal to 23 is the *same set of numbers* that are NOT greater than 23!

Answer ([back](#)) **Correct answer is C**

Practice AP Question

A taxi cab is willing to drive at most 50 miles in a trip, and charges \$2 per mile, plus a surcharge of \$10 on trips between 20 miles and 40 miles, and a \$15 surcharge on trips over 40 miles.

Two procedures are defined below:

- The `taxiDrive` procedure takes a parameter, `distance`, and returns `true` if the taxi will drive that far and `false` if it will not.
- The `taxiCost` procedure also takes a parameter, `distance`, and returns the cost of the trip.

```
PROCEDURE taxiDrive(distance)
{
  IF (distance > 50)
  {
    RETURN false
  }
  ELSE
  {
    RETURN true
  }
}
```

```
PROCEDURE taxiCost(distance)
{
  IF (distance < 20)
  {
    RETURN (distance * 2)
  }
  ELSE
  {
    IF ((distance ≥ 20) AND (distance ≤ 40))
    {
      RETURN (distance * 2 + 10)
    }
    ELSE
    {
      RETURN (distance * 2 + 15)
    }
  }
}
```

Determine what is displayed when the code segment below is executed:

```
trip ← 35
totalCost ← 0
IF (taxiDrive(trip))    taxiDrive(35) returns true
{
  totalCost ← taxiCost(trip)    taxiCost(35) returns 80
}
DISPLAY("The cost of the trip is " + totalCost)
```

- A. The cost of the trip is 0.
- B. The cost of the trip is 70.
- C. The cost of the trip is 80.**
- D. The cost of the trip is 85.

Explanation:

```
trip ← 35
totalCost ← 0
IF (taxiDrive(35))  ← 35 is the argument to the taxiDrive procedure, and it
                    returns true because the number is not greater than 50
{
    totalCost ← taxiCost(35)  ← so this code runs because the if condition was
                              true, and 35 gets passed as the argument to
                              the taxiCost procedure, which returns 80 since
                              (35 ≥ 20) AND (35 ≤ 40) is true (because both
                              parts are true), so RETURN (35 * 2 + 10)
                              returns 80 from the procedure, which gets
                              assigned to totalCost
}
DISPLAY("The cost of the trip is " + 80)  ← totalCost = 80
```

([back](#))

Answers ([back](#))

#	Expression	Ans.	#	Expression	Ans.
1	10 MOD 1 = 10 divided by 1 = 10 remainder 0	0	11	10 MOD 15 = 10 divided by 15 = 0 remainder 10 (15 doesn't fit into 10 at all!)	10
2	10 MOD 2 = 10 divided by 2 = 5 remainder 0	0	12	10 MOD 30 = 10 divided by 30 = 0 remainder 10 (30 doesn't fit into 10 at all!)	10
3	10 MOD 3 = 10 divided by 3 = 3 remainder 1	1	13	18 MOD 5 = 18 divided by 5 = 3 remainder 3	3
4	10 MOD 4 = 10 divided by 4 = 2 remainder 2	2	14	5 MOD 18 = 5 divided by 18 = 0 remainder 5 (18 doesn't fit into 5 at all!)	5
5	10 MOD 5 = 10 divided by 5 = 2 remainder 0	0	15	(3 MOD 2) + (6 MOD 4) = (the remainder of 3 div 2) + (the remainder of 6 div 4) = 1 + 2 = 3	3
6	10 MOD 6 = 10 divided by 6 = 1 remainder 4	4	16	15 MOD (12 MOD 5) = 15 MOD (the remainder of 12 divided by 5) = 15 MOD 2 = 15 divided by 2 = 7 remainder 1	1
7	10 MOD 7 = 10 divided by 7 = 1 remainder 3	3	17	(10 MOD 2) = 0 10 mod 2 = 0 and so this expression is TRUE	True
8	10 MOD 8 = 10 divided by 8 = 1 remainder 2	2	18	(14 MOD 4) = (13 MOD 3) 14 mod 4 = 2 and 13 mod 3 = 1 so they are NOT equal so this expression is FALSE	False
9	10 MOD 9 = 10 divided by 9 = 1 remainder 1	1	19	(25 MOD 13) = (12 MOD 24) 25 mod 13 = 12 (25 div 13 = 1 remainder 12) and 12 mod 24 = 12 (12 div 24 = 0 remainder 12) and 12 = 12, so this expression is TRUE	True
10	10 MOD 10 = 10 divided by 10 = 1 remainder 0	0	20	(12 MOD 2) > (11 MOD 2) 12 mod 2 = 0 and 11 mod 2 = 1, and 0 is not greater than 1 so this expression is FALSE	False

Answer ([back](#))

Determine what is displayed when this code segment below is executed (**reminder!** `[]` means an empty list):

```
finalList ← []
nums ← [4, 15, 10, 9, 20, 1, 17, 13]
FOR EACH num IN nums
{
  IF ((num MOD 3) = 1)
  {
    APPEND(finalList, num)
  }
}
DISPLAY(finalList)
```

What gets displayed should be a list!
Write it using `[]` notation:

[4, 10, 1, 13]

In the first line, `finalList` gets set to an empty list, then iterates through the list with a `FOR EACH` loop, adding to the list *only those numbers that, when “modded” with 3, equals to 1*:

4 divided by 3 = 1 **Remainder 1**, so $4 \text{ MOD } 3 = 1$ ✓ → add 4 to `finalList`

15 divided by 3 = 5 **Remainder 0**, so $15 \text{ MOD } 3 \neq 1$ ✗

10 divided by 3 = 3 **Remainder 1**, so $10 \text{ MOD } 3 = 1$ ✓ → add 10 to `finalList`

9 divided by 3 = 3 **Remainder 0**, so $9 \text{ MOD } 3 \neq 1$ ✗

20 divided by 3 = 6 **Remainder 2**, so $20 \text{ MOD } 3 \neq 1$ ✗

1 divided by 3 = 0 **Remainder 1**, so $1 \text{ MOD } 3 = 1$ ✓ → add 1 to `finalList`

17 divided by 3 = 5 **Remainder 2**, so $17 \text{ MOD } 3 \neq 1$ ✗

13 divided by 3 = 4 **Remainder 1**, so $13 \text{ MOD } 3 = 1$ ✓ → add 13 to `finalList`

`finalList = [4, 10, 1, 13]`

Recall that `APPEND` adds the value to the **end** of the list:

Text:
`APPEND(aList, value)`

Block:

`APPEND` `aList, value`

The length of `aList` is increased by 1, and `value` is placed at the end of `aList`.

Check ([back](#))

When working with **mathematical operations**, the same **order of operations** (PEMDAS) from math apply!

What would the expression $(4 * 10) - 2$ evaluate to?

$$40 - 2 = \mathbf{38}$$

What would the expression $4 * (10 - 2)$ evaluate to?

$$4 * 8 = \mathbf{32}$$

What would the expression $12 + 8 / 4$ evaluate to?

$$12 + 2 = \mathbf{14}$$

How about $2 * 3 + 4 / 2 - 1$?

$$6 + 2 - 1 = 8 - 1 = \mathbf{7}$$

What would $30 - \mathbf{14} \bmod \mathbf{5} * 7 \bmod 4$ evaluate to?

(MOD has the *same* precedence as * and /, i.e. MOD, *, / left to right)

$$\begin{aligned} &= 30 - \mathbf{4} * 7 \bmod 4 \\ &= 30 - 28 \bmod 4 \\ &= 30 - 0 \\ &= \mathbf{30} \end{aligned}$$

Answer ([back](#))

Correct Answer: D

AP EXAM PRACTICE QUESTION #1

The following incomplete code fragment was designed to test if `number` is odd:

```
IF (<MISSING CONDITION>)  
{  
    DISPLAY ("The number is odd!")  
}
```

Which of the following can be used in place of `<MISSING CONDITION>` ?

X the correct answer choice.

	A. <code>number MOD 1 = 0</code>
	B. <code>number MOD 1 = 1</code>
	C. <code>number MOD 2 = 0</code>
X	D. <code>number MOD 2 = 1</code> Testing for “odd” requires that the number is not divisible by 2. In other words, there is a remainder of 1 when the number is divided by 2; this is what <code>number MOD 2 = 1</code> is testing for!

Answer ([back](#))

Correct Answer: B

AP EXAM PRACTICE QUESTION #2

Consider the following code segment:

```
i ← 100
repeat until i = 0
{
    <MISSING CODE>
}
```

Which of the following replacements for <MISSING CODE> will cause an infinite loop?

X the correct answer choice.

	<p>A. $i \leftarrow i \text{ MOD } 2$</p> <p>With $i = 100$, in the first iteration of the loop, i gets set to $i \text{ mod } 2$ which equals $100 \text{ mod } 2$ which equals 0. The loop then ends because the repeat until condition ($i = 0$) condition is true.</p>
X	<p>B. $i \leftarrow i \text{ MOD } 3$</p> <p>With $i = 100$, in the first iteration of the loop, i gets set to $i \text{ mod } 3$ which equals $100 \text{ mod } 3$ which equals 1. Now with $i = 1$, the loop then repeats forever because the line $i \leftarrow 1 \text{ MOD } 3$ sets i to $1 \text{ MOD } 3$, which is 1 again! <i>This goes on forever!</i></p>
	<p>C. $i \leftarrow i \text{ MOD } 4$</p> <p>With $i = 100$, in the first iteration of the loop, i gets set to $i \text{ mod } 4$ which equals $100 \text{ mod } 4$ which equals 0. The loop then ends because the repeat until condition ($i = 0$) condition is true.</p>
	<p>D. $i \leftarrow i \text{ MOD } 5$</p> <p>With $i = 100$, in the first iteration of the loop, i gets set to $i \text{ mod } 5$ which equals $100 \text{ mod } 5$ which equals 0. The loop then ends because the repeat until condition ($i = 0$) condition is true.</p>

Answer ([back](#))

Correct Answer: C

AP EXAM PRACTICE QUESTION #3

A local coffee shop started passing out punch cards that, for each time you purchase a coffee, they punch a hole in the card. When you get six punched holes, you get a free coffee! A software programmer wanted to write a program to display a message to the customer on the cash register when they have earned their free coffee. Which of the following segments of code could be used to display this message at the right time?

X the correct answer choice.

	<p>A.</p> <pre>IF (punchedHoles / 6) = 0 { DISPLAY("Free coffee!") }</pre>
	<p>B.</p> <pre>IF NOT((punchedHoles / 6) = 0) { DISPLAY("Free coffee!") }</pre>
X	<p>C.</p> <pre>IF (punchedHoles MOD 6) = 0 { DISPLAY("Free coffee!") }</pre> <p>The expression <code>(punchedHoles MOD 6) = 0</code> evaluates to true when punchedHoles is a multiple of 6, and so this code segment would display the “free coffee!” message at the correct time.</p>
	<p>D.</p> <pre>IF NOT((punchedHoles MOD 6) = 0) { DISPLAY("Free coffee!") }</pre>

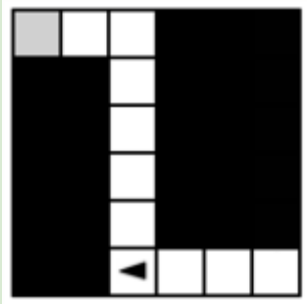
Answer ([back](#))

Correct Answer: B

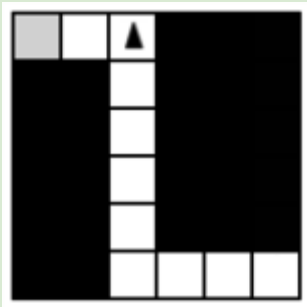
X

(B) TurnAndMove (1, 3)
TurnAndMove (3, 5)
TurnAndMove (1, 2)

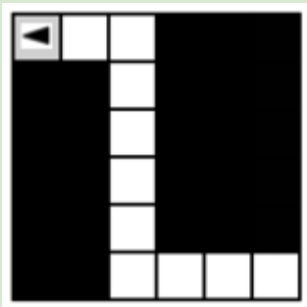
TurnAndMove (1, 3) will turn the robot **left 1 time**, then move it **forward 3 times**:



TurnAndMove (3, 5) will turn the robot **left 3 times** (which has the *same* effect as turning right 1 time), then move it **forward 5 times**:



TurnAndMove (1, 2) will turn the robot **left 1 time**, then move it **forward 2 times**, which takes it to the gray square:



Answer ([back](#))

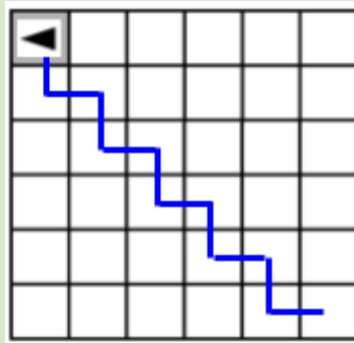
Correct Answers: B & D

AP EXAM PRACTICE QUESTION #2

(A) REPEAT 5 TIMES
{
 MOVE_FORWARD ()
 ROTATE_RIGHT ()
}
REPEAT 5 TIMES
{
 MOVE_FORWARD ()
 ROTATE_LEFT ()
}

X

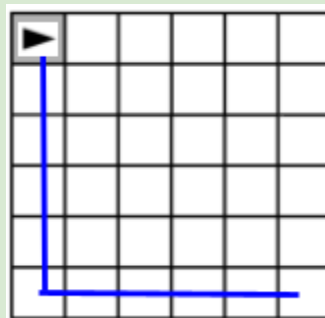
(B) REPEAT 5 TIMES
{
 MOVE_FORWARD ()
 ROTATE_RIGHT ()
 MOVE_FORWARD ()
 ROTATE_LEFT ()
}



(C) REPEAT 5 TIMES
{
 REPEAT 2 TIMES
 {
 MOVE_FORWARD ()
 }
 ROTATE_RIGHT ()
}

X

(D) REPEAT 2 TIMES
{
 REPEAT 5 TIMES
 {
 MOVE_FORWARD ()
 }
 ROTATE_RIGHT ()
}

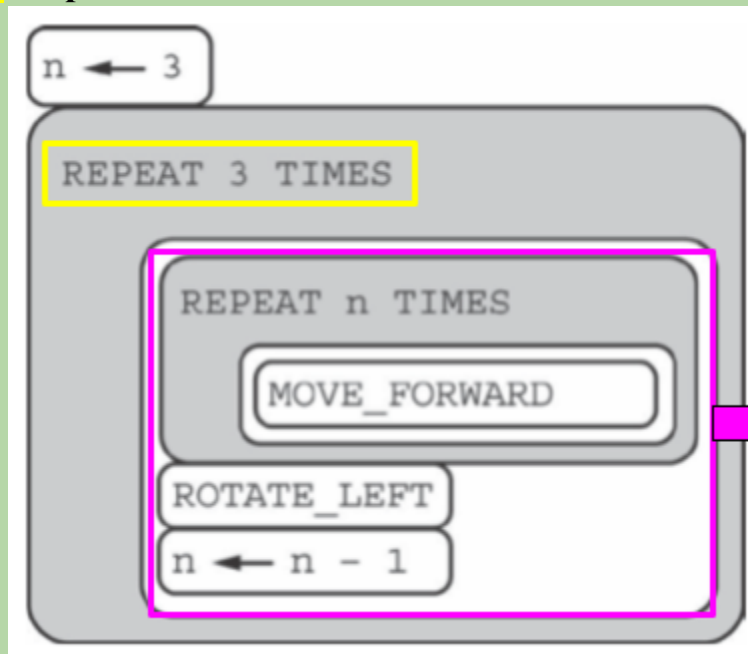


Note that the robot is pointing to the right, but that's OK since it is in the gray square!

Answer ([back](#))

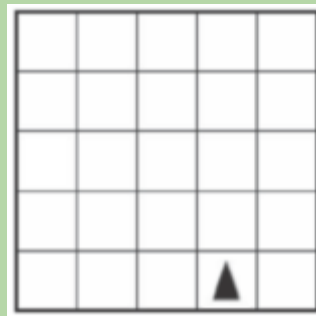
Correct Answer: A

X (A) Explanation:

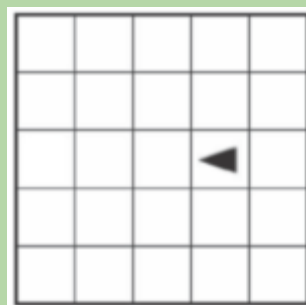


This code gets run **three** times because of the "Repeat 3 Times" outside loop

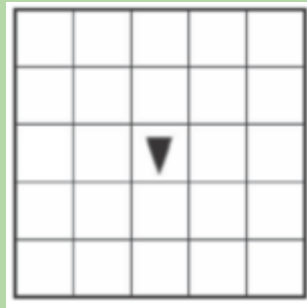
The **first** time through, $n = 3$, so the "robot" moves forward **three** times (because of "REPEAT n times"), then rotates **left**, and n is decreased by 1 down to 2. The "robot" will be here:



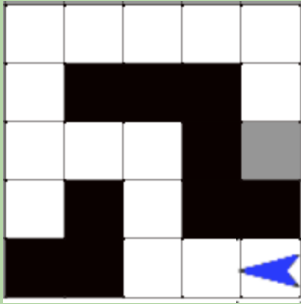
The **second** time through, $n = 2$, so the arrow moves forward **two** times (because of "REPEAT n times"), then rotates **left**, and n is decreased by 1 down to 1. The "robot" will be here:



The **third and final** time through, $n = 1$, so the arrow moves forward **one** time (because of “REPEAT n times”), then rotates **left**, and n is decreased by 1 down to 0. The “robot” will be here, which is its **final location and answer choice A**:



A.



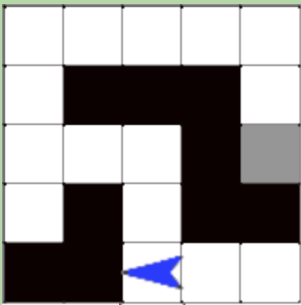
`CAN_MOVE(right)` = false (from the perspective of the robot, the right is a black space = boundary!)

`CAN_MOVE(left)` = false (from the perspective of the robot, the left is the edge of the maze = boundary!)

`CAN_MOVE(forward)` = true (the white space to the left of the robot is FORWARD since robot is pointing left)

`CAN_MOVE(backward)` = false

B.



`CAN_MOVE(right)` = true

`CAN_MOVE(left)` = false

`CAN_MOVE(forward)` = false

`CAN_MOVE(backward)` = true

C.



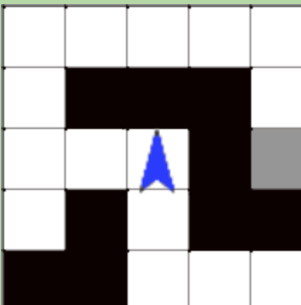
`CAN_MOVE(right)` = true

`CAN_MOVE(left)` = false

`CAN_MOVE(forward)` = true

`CAN_MOVE(backward)` = false

D.



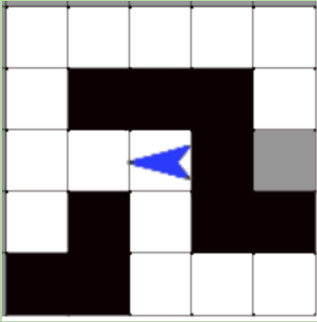
`CAN_MOVE(right)` = false

`CAN_MOVE(left)` = true

`CAN_MOVE(forward)` = false

`CAN_MOVE(backward)` = true

E.

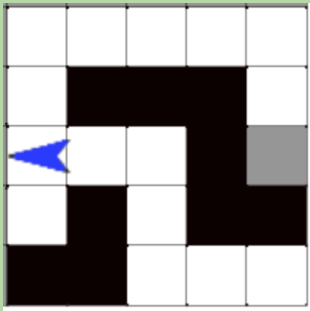
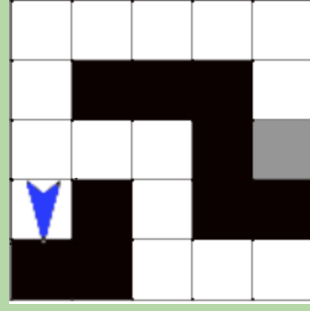
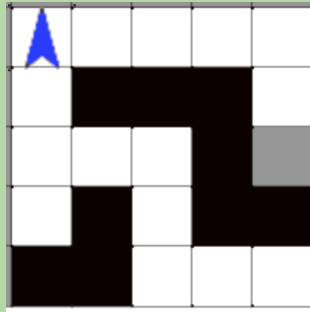
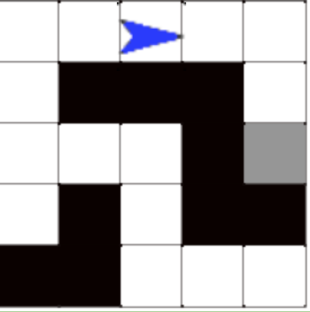


`CAN_MOVE(right) = false`

`CAN_MOVE(left) = true`

`CAN_MOVE(forward) = true`

`CAN_MOVE(backward) = false`

<p>F.</p> 	<p>CAN_MOVE(right) = true</p> <p>CAN_MOVE(left) = true</p> <p>CAN_MOVE(forward) = false</p> <p>CAN_MOVE(backward) = true</p>
<p>G.</p> 	<p>Careful then the robot is facing down!</p> <p>CAN_MOVE(right) = false</p> <p>CAN_MOVE(left) = false</p> <p>CAN_MOVE(forward) = false</p> <p>CAN_MOVE(backward) = true</p>
<p>H.</p> 	<p>CAN_MOVE(right) = true</p> <p>CAN_MOVE(left) = false</p> <p>CAN_MOVE(forward) = false</p> <p>CAN_MOVE(backward) = true</p>
<p>I.</p> 	<p>CAN_MOVE(right) = false</p> <p>CAN_MOVE(left) = false</p> <p>CAN_MOVE(forward) = true</p> <p>CAN_MOVE(backward) = true</p>

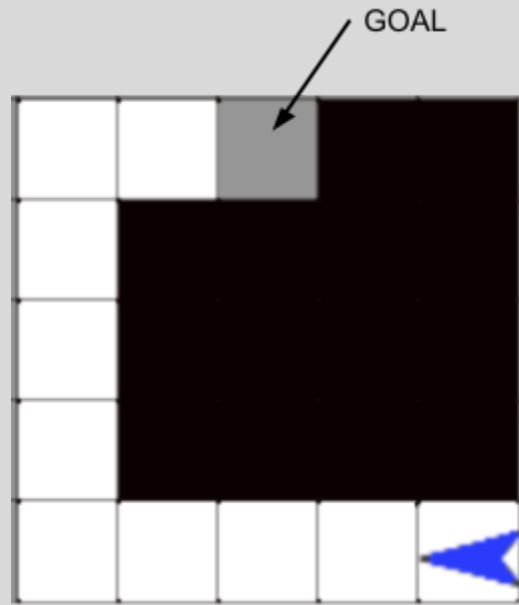
<p>J.</p>	<pre> CAN_MOVE(right) = true CAN_MOVE(left) = false CAN_MOVE(forward) = false CAN_MOVE(backward) = true </pre>
<p>K.</p>	<p>Careful then the robot is facing down!</p> <pre> CAN_MOVE(right) = true CAN_MOVE(left) = false CAN_MOVE(forward) = true CAN_MOVE(backward) = false </pre>
<p>L.</p>	<pre> CAN_MOVE(right) = false CAN_MOVE(left) = false CAN_MOVE(forward) = false CAN_MOVE(backward) = true </pre> <p>THE GOAL (gray space) HAS BEEN REACHED!</p>

([back](#))

Answer ([back](#))

YES, it *would* get the robot to the goal

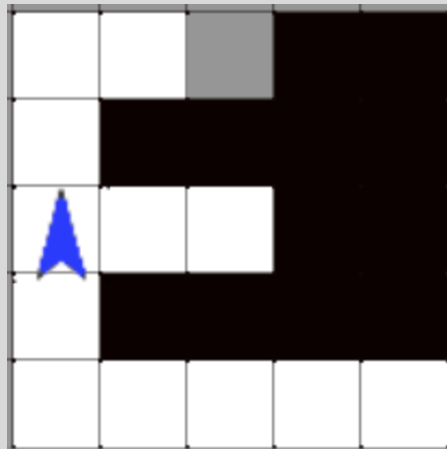
```
REPEAT UNTIL (goal_reached)
{
  IF (CAN_MOVE(forward))
  {
    MOVE_FORWARD()
  }
  IF (CAN_MOVE(left))
  {
    ROTATE_LEFT()
  }
  IF (CAN_MOVE(right))
  {
    ROTATE_RIGHT()
  }
}
```



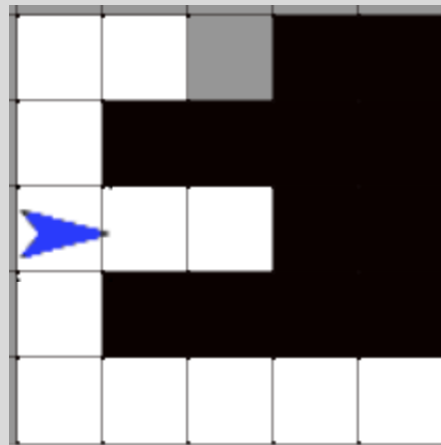
The “repeat until” block repeats until the robot gets to the gray square. Each time through, it tests which direction it can move, and if possible, moves or rotates. The robot will keep moving forward *until* it reaches the bottom left corner. When it gets to the bottom left corner, **can move forward** is now **false** but **can move right** is **true**, so it rotates right, and then it continues to move forward until the top left corner, when it rotates right again. Finally, it moves forward until the goal is reached!

Answer ([back](#))

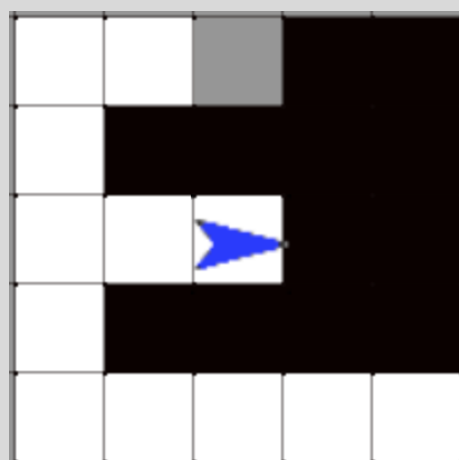
NO! It will NOT. The problem is when the robot gets to **this space**:



When it gets here by moving forward, the next line in the code, **can move left** is **false**, but the following line, **can move right** is **true**, so the sprite turns right:



The loop then iterates back to the beginning, and the next line of code, **can move forward** is **true**, so it then proceeds to move forward until it gets here:



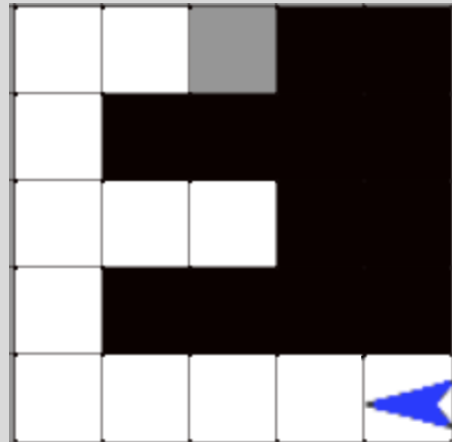
And now it's **stuck forever** because it can't move in **any** direction! So the loop is now an infinite loop since the goal will never be reached.

Answer ([back](#))

If the following code segment was executed, in which the `CAN_MOVE(left)` and `CAN_MOVE(right)` conditions have been **swapped** from the code in the previous problem, would the robot reach the goal (gray square)?

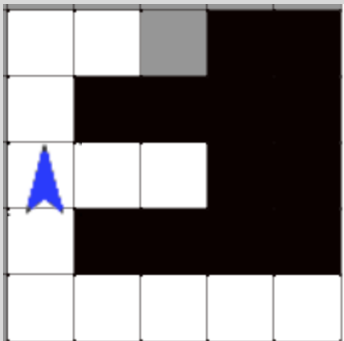
YES! It would

```
REPEAT UNTIL (goal_reached)
{
  IF (CAN_MOVE(forward))
  {
    MOVE_FORWARD()
  }
  IF (CAN_MOVE(right))
  {
    ROTATE_RIGHT()
  }
  IF (CAN_MOVE(left))
  {
    ROTATE_LEFT()
  }
}
```

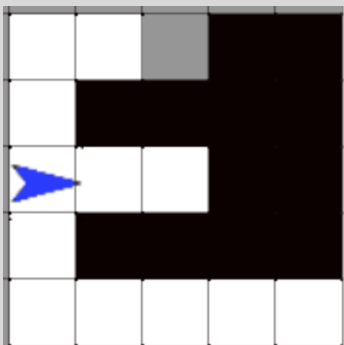


EXPLANATION

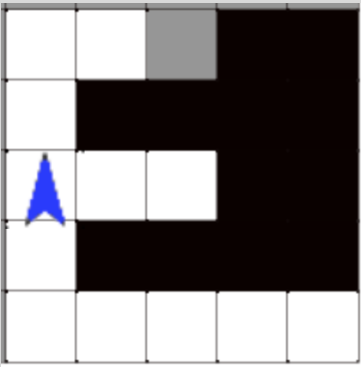
When it gets here by moving forward:



The next line in the code, **can move right** is **true**, so it rotates right:



But the **following** line of code, **can move left**, now also is **true**, *given the robot's orientation*, so the sprite turns back left:



The loop then iterates back to the beginning, and the next line of code, **can move forward** is **true**, so it then proceeds to move forward and follow the maze to the end!

Note how swapping those two lines of code changes the result!

([back](#))

Answer ([back](#))

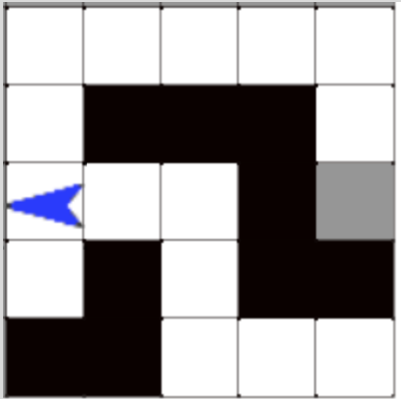
A. Code segment A, only

B. Code segment B, only

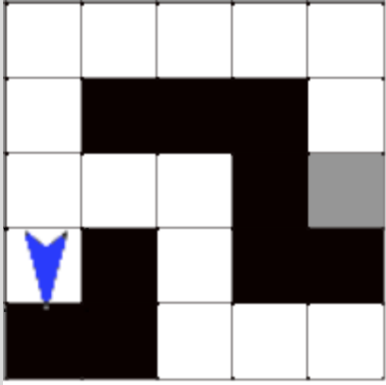
C. Both code segments A and B

D. Neither code segment A nor B

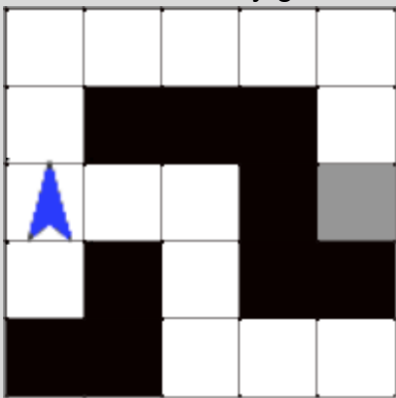
Only code segment A would work! In code segment B, when the Robot gets here:



It would rotate **left** instead of **right** because the **can move left** appears in the code **before** the **can move right**. This then leads it to become **stuck** forever:



In code segment **A**, however, the **can move right** appears *first* in the code, so it rotates right, which allows to ultimately get to the goal:



Answer ([back](#))

Correct Answer: D

X (D) Explanation

Given this code:

```
REPEAT UNTIL (GoalReached ())
{
    IF (CAN_MOVE (forward))
    {
        MOVE_FORWARD ()
    }
    IF (CAN_MOVE (right))
    {
        ROTATE_RIGHT ()
    }
    IF (CAN_MOVE (left))
    {
        ROTATE_LEFT ()
    }
}
```

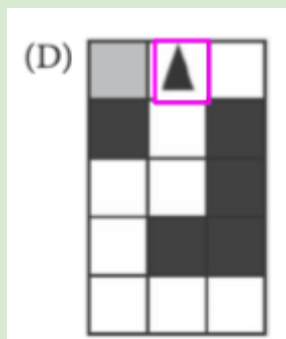
The robot will successfully complete all mazes **except** D.

```
IF (CAN_MOVE (right))
{
    ROTATE_RIGHT ()
}
```

This is because `ROTATE_RIGHT ()` appears in the code **before**

```
IF (CAN_MOVE (left))
{
    ROTATE_LEFT ()
}
```

so when the robot gets to this position:

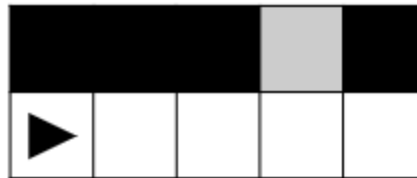


It will rotate **right instead of left**, and then get stuck in the top right corner and never reach the Goal.

Answer ([back](#))

Correct Answers: A & D

The grid below contains a robot represented as a triangle, initially facing up. The robot can move into a white or gray square but cannot move into a black region.



Which two of the following code segments can be used to move the robot to the gray square?

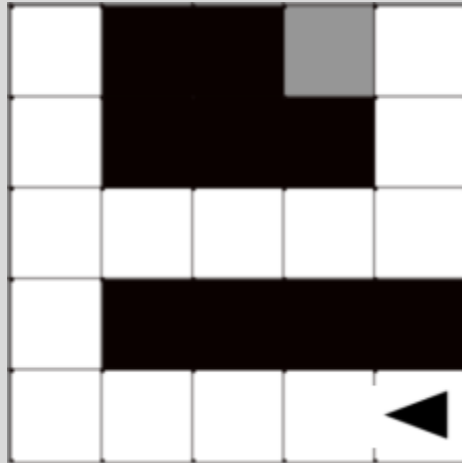
Select two answers.

X the TWO correct answer choices.

X	(A) REPEAT 3 TIMES { MOVE_FORWARD () } ROTATE_LEFT () MOVE_FORWARD ()
	(B) REPEAT 4 TIMES { MOVE_FORWARD () } ROTATE_LEFT () MOVE_FORWARD ()
	(C) IF (CAN_MOVE (left)) { ROTATE_LEFT () } MOVE_FORWARD () Note that this answer choice would have been correct if it had been inside a “Repeat Until GoalReached()” loop.
X	(D) REPEAT UNTIL (CAN_MOVE (left)) { MOVE_FORWARD () } ROTATE_LEFT () MOVE_FORWARD ()

Answer ([back](#))

Correct Answer: B



X (B) II, only

When the robot moves forward to get to this point:

```
IF (CAN_MOVE (left))  
{  
    ROTATE_LEFT ()  
}
```

The next line of code is `ROTATE_LEFT ();` in this position, the robot can't move left, so no rotation happens.

```
IF (CAN_MOVE (right))  
{  
    ROTATE_RIGHT ()  
}
```

The *next* line of code is `ROTATE_RIGHT ();` in this position, the robot **can** move right, so it rotates right.

Then,

