# Conditions and Logic

Computer programs make decisions based on logic: if some condition applies, do something, otherwise, do something else.

## Content Learning Objectives

*After completing this activity, students should be able to:*

- Evaluate boolean expressions with comparison operators (<, >, <=, >=, ==, !=).
- Explain the syntax and meaning of `if`/`else` statements and indented blocks.
- Evaluate boolean expressions that involve comparisons with `and`, `or`, and `not`.

## Process Skill Goals

*During the activity, students should make progress toward:*

- Evaluating complex logic expressions based on operator precedence. (Critical Thinking)

## Facilitation Notes

Model 1 is quick and straightforward; you may not need to report out at all. Just keep an eye on each team's responses and give individual help as needed.

On Model 2, have teams share their answer for #9. Ask other teams if they tried additional experiments in the Python Shell to figure out the indenting rules of Python. You may want to demonstrate (on the projector) what happens if you forget the colon. It might also be helpful to show an if-statement with more than one line in the body.

During Model 3, explain that the variables $p$ and $q$ are often used to represent logic values in discrete math. Explain that "`not`" is a unary operator, and that "`and`" and "`or`" are binary operators. Double check each team's truth table before they complete the remaining questions.

Have each team write their answers to #24 and #25 on the board. Ideally you will have a variety of solutions, some of which are logically equivalent. Discuss the correct solutions to these questions, and reinforce intuition about logic.

As a wrap-up discussion, step through the last print statement of Model 3 using a debugger. Have the students check their work as you demonstrate the order of operations and the result of each expression.

# Model 1 Comparison Operators

In Python, a comparison (e.g., `100 < 200`) will yield a ***Boolean*** value of either `True` or `False`. Most data types (including `int`, `float`, `str`, `list`, and `tuple`) can be compared using the following operators:

| Operator | Meaning |
|---|---|
| < | less than |
| <= | less than or equal |
| > | greater than |
| >= | greater than or equal |
| == | equal |
| != | not equal |

Type the following code, one line at a time, into a Python Shell. Record the output for each line (if any) in the second column.

| Python code | Shell output |
|---|---|
| `type(True)` | <class 'bool'> |
| `type(true)` | NameError |
| `type(3 < 4)` | <class 'bool'> |
| `print(3 < 4)` | True |
| `three = 3` | |
| `four = 4` | |
| `print(three == four)` | False |
| `check = three > four` | |
| `print(check)` | False |
| `type(check)` | <class 'bool'> |
| `print(three = four)` | TypeError |
| `three = four` | |
| `print(three == four)` | True |

## Questions (10 min) Start time: _____

1. What is the name of the data type for Boolean values? bool

2. Do the words `True` and `False` need to be capitalized? Explain how you know.

Yes, because `type(true)` resulted in NameError: name 'true' is not defined.

3. For each of the following terms, identify examples from the table in Model 1:

   a) Boolean variables: check

   b) Boolean operators: <, ==, >

   c) Boolean expressions: 3 < 4, three == four, three > four

4. Explain why the same expression `three == four` had two different results.

The two variables were initially different values, so the first comparison was `False`. But later on, the value of `four` was assigned to `three`, so the second comparison was `True`.

5. What is the difference between the = operator and the == operator?

The = operator assigns a value to a variable, and the == operator compares two values.

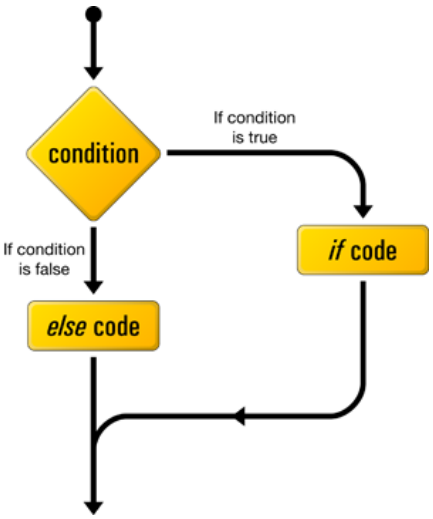
6. Write a Boolean expression that uses the `!=` operator and evaluates to `False`.

5 != 5

## Model 2 `if`/`else` Statements

An `if` statement makes it possible to control what code will be executed in a program, based on a condition. For example:

```
number = int(input("Enter an integer: "))
if number < 0:
    print(number, "is negative")
else:
    print(number, "is a fine number")
print("Until next time...")
```




Python uses ***indentation*** to define the structure of programs. The line indented under the `if` statement is executed only when `number < 0` is `True`. Likewise, the line indented under the `else` statement is executed only when `number < 0` is `False`. The flowchart on the right illustrates this behavior.

## Questions (15 min) Start time: _____

**7.** What is the Boolean expression in Model 2?

number < 0

**8.** Enter this short program into a Python Editor. What is the output when the user enters the number 5? What is the output when the user enters the number -5?

```
5 is a fine number              -5 is negative
Until next time...              Until next time...
```

**9.** After an if-condition, what syntax differentiates between (1) statements that are executed based on the condition and (2) statements that are always executed?

The indentation; statements that are indented under the if are based on the condition, and statements indented at the same level (later in the program) are always executed.

**10.** Enter the line ␣␣␣␣`print("Hello")` into a Python Editor (where ␣ is a space), save the file as `hello.py`, and run the program. What happens if you indent code inconsistently?

SyntaxError: unexpected indent

**11.** Based on the program in Model 2, what must each line preceding an indented block of code end with?

A colon.

**12.** Write an `if` statement that first determines whether `number` is even or odd, and then prints the message `"(number) is even"` or `"(number) is odd"`. (Hint: use the `%` operator.)

```
if number % 2 == 0:
    print(number, "is even")
else:
    print(number, "is odd")
```

**13.** Does an `if` statement always need to be followed by an `else` statement? Why or why not? Give an example.

No; you can have an if statement without an else. For example, you could determine that a number is even and print a message, without printing a different message if it's odd.

# Model 3 Boolean Operations

Expressions may include Boolean operators to implement basic logic. If all three operators appear in the same expression, Python will evaluate `not` first, then `and`, and finally `or`. If there are multiple of the same operator, they are evaluated from left to right.

**Do not type anything yet! Read the questions first!**

| Python code | Predicted output | Actual output |
|---|---|---|
| `print(a < b and b < c)` | | True |
| `print(a < b or b < c)` | | True |
| `print(a < b and b > c)` | | False |
| `print(a < b or b > c)` | | True |
| `print(not a < b)` | | False |
| `print(a > b or not a > c and b > c)` | | False |

## Questions (20 min)

**Start time:** _____

**14.** What data type is the result of `a < b`? What data type is the result of `a < b and b < c`?

The type of each is `bool`; both are Boolean expressions.

**15.** Predict the output of each print statement, based on the variables `a = 3`, `b = 4`, and `c = 5`. Then execute each line in a Python Shell to check your work.

**16.** Based on the variables in #15, what is the value of a < b? What is the value of b < c?

They are both true.

**17.** If two `True` Boolean expressions are combined using the `and` operator, what is the resulting Boolean value?

True and True is True.

**18.** Using the variables defined in #15, write an expression that will combine two `False` Boolean expressions using the `or` operator. Check your work using a Python Shell.

a > b or a > c

**19**. Assuming `P` and `Q` each represent a Boolean expression that evaluates to the Boolean value indicated, complete the following table. Compare your team's answers with another team's, and resolve any inconsistencies.

| P | Q | P and Q | P or Q |
|---|---|---|---|
| False | False | False | False |
| False | True | False | True |
| True | False | False | True |
| True | True | True | True |

**20**. Assume that two Boolean expressions are combined using the `and` operator. If the value of the first expression is `False`, is it necessary to determine the value of the second expression? Explain why or why not.

It is unnecessary, because "false and anything" is false.

**21**. Assume that two Boolean expressions are combined using the `or` operator. If the value of the first expression is `True`, is it necessary to determine the value of the second expression? Explain why or why not.

It is unnecessary, because "true or anything" is true.

**22**. Examine the last row of the table in #15. Evaluate the Boolean expression following the order of precedence rules explained in Model 3. Show your work by rewriting the line at each step and replacing portions with either `True` or `False`.

```
a > b or not a > c and b > c
```

```
False or not a > c and b > c
False or not False and b > c
   False or True and b > c
   False or True and False
       False or False
           False
```

**23**. Suppose you wanted to execute the statement `sum = x + y` only when both `x` and `y` are positive. Determine the appropriate operators, and write a single Boolean expression for the if-condition.

x > 0 and y > 0

**24**. Rewrite the expression from #23 using the `not` operator. Your answer should yield the same result as in #23, not the opposite. Describe in words what the new expression means.

not (x <= 0 or y <= 0)
In other words, "both x and y are positive" is equivalent to "neither x nor y is negative/zero".

**25**. Suppose that your team needs to execute the statement `sum = x + y` except when both `x` and `y` are positive. Write a Boolean expression for this condition. How is it different from the previous question?

not (x > 0 and y > 0)
To represent "except when" logic, we simply negate the original condition. The previous question negated each of the operators as well, which is known as De Morgan's law.