CONDITIONALS (LECTURE)

RECAP: CONDITIONS AS BOOLEAN EXPRESSIONS

Boolean expressions evaluate to `bool` values, that is to say, either `True` or `False`.

```
3 < 4 and 9 == (81 / 9)                    # always True
not True and True or False and not False   # always False
```

To write these expressions we use a combination of **values** and **operators**

THE BOOLEAN EXPRESSION TOOLKIT

Relational Operators:

OPERATOR/METHOD	INPUT TYPES	DESCRIPTION
< / <=	int, float, str	less than / less than or equal to
> / >=	int, float, str	greater than / greater than or equal to
== / !=	int, float, str	equal to / not equal to

THE BOOLEAN EXPRESSION TOOLKIT

Logical Operators:

OPERATOR/METHOD	INPUT TYPES	DESCRIPTION
and	bool	evaluates to **True** only if both inputs are **True**
or	bool	evaluates to **True** as long as at least one input is **True**
not	bool	negates a single *bool* value to its opposite

We can write boolean expressions that contain variables

```
x % 3 == 2 and x > 5                    # not always True or False!
```

This expression's value changes based on the value of `x`!

L1: Can you think of a value of `x` that would cause the expression to evaluate to `True`? What about `False`?

THE IN KEYWORD

In its simplest usage, it checks to see if `something` is within `something else`.

It evaluates to either `True` or `False`. As an example:

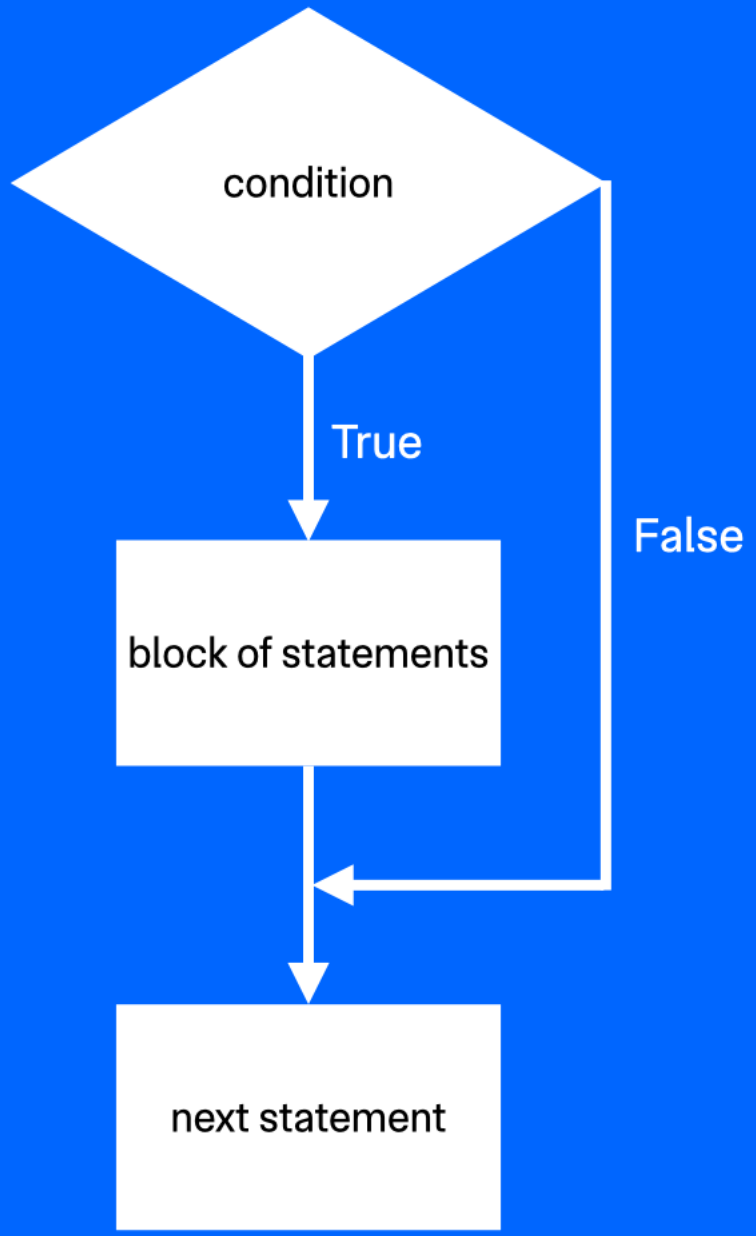
```
result = "pressure" in "under_pressure"
print(result) # prints True!!

result = "Maracuyá" in "Mar"
print(result) # prints False!
```

We will eventually see more complex usage; but this is ok for now. :)

RECAP: IF

The `if` statement allows us to specify a portions of our program that should be run **only in the case that** a certain condition is met.

```
if my_boolean_expression:
        statement_one
        statement_two
        ...
        statement_last
```



RECAP: CONTROL FLOW & IF

- Test the condition...
  - **if** it is `True`, execute the block of statements
  - otherwise, proceed to the next statement.

EXAMPLE: IF

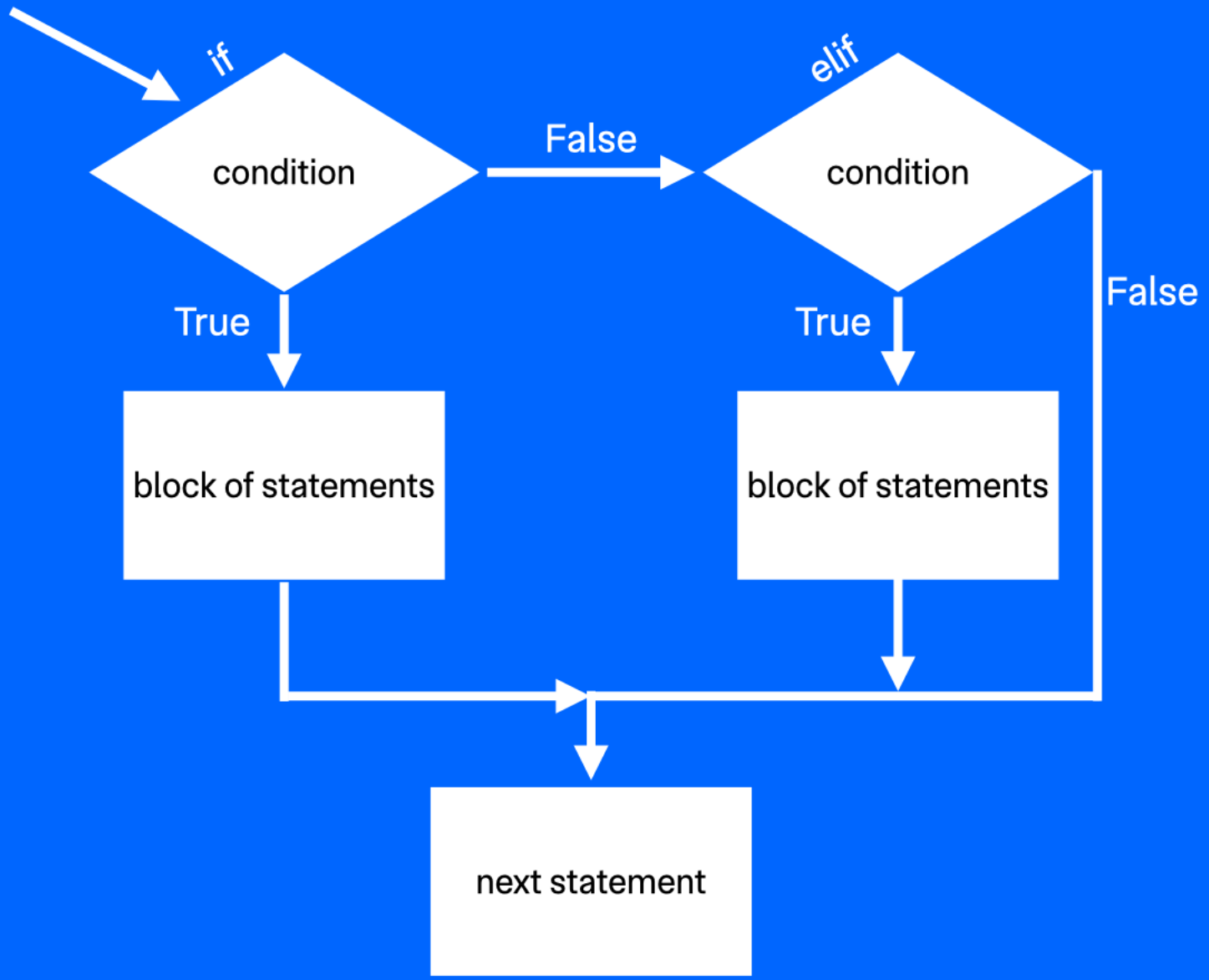
```
# A program for leaving the house
grab phone
grab wallet
grab keys

look outside
if it_is_raining:
        grab an umbrella

walk outside
```

RECAP: ELIF

`elif` allows you to specify an *alternative condition* that is be tested *only when all previous conditions were* `False`.

The `elif` syntax:

```
if first_boolean_expression:
        statement_one
        statement_two
        ...
elif alternative_boolean_expression:
        statement_a
        statement_b
        ...
```



RECAP: ELIF

`if` and `elif` statements represent *mutually exclusive* choices: we may execute the body of one, the other, or neither, but *never both*.

EXAMPLE: ELIF

```
# A program for leaving the house
grab phone
grab wallet
grab keys

look outside
if it_is_raining:
    grab an umbrella
elif it_is_snowing: # NOTE: can it be both rain and snow?? (NO)
    grab my boots

walk outside
```

RECAP: ELSE

The `else` keyword allows us to define a body of statements that will be run if all previous conditions (`if` and `elif`) were not met.

```
if first_boolean_expression:
    block_one
elif alternative_boolean_expression:
    block_two
# optionally many elif statements provided here...
else:
    block_three
```

Look: no new condition provided!

EXAMPLE: ELSE

```
# A program for leaving the house
grab phone
grab wallet
grab keys

look outside
if it_is_raining:
    grab an umbrella
elif it_is_snowing:
    grab my boots
else: # NOTE: this means not raining NOR snowing
    grab a book

walk outside
```

RECAP: ALL CONDITIONALS

Recipe for any conditional:

1. Always start with an `if`. Each `if` comes with a boolean expression to test. This expression is always tested.
2. As many `elif` statements as desired. Each comes with a boolean expression. Each expression only tested if all previous are `False`.
3. An `else` statement, or not. No boolean expression provided. Body executed if all previous expressions are `False`.

Let's do some practice activities to build our understanding of boolean expressions and conditionals!

Just like last class, we're going to be working with programs in a sort of loose Python-y thing to describe real actions, but the rules are all the same.

We're also going to go outside the classroom. Before that, let's explain the domain of our variables and the actions we can take:

OUR DOMAIN: POMPOMS 😄

- Color: ⬜, 🟥, 🩷, 🟨, 🟩, 🩵 (l), 🟦 (d), 🟪
- Sparkles: ✨, 🚫
- Size: S, M, L

WHAT ACTIONS CAN WE TAKE?

`put_in_bowl(bowl_label)`

`put_in_bag(bag_label)`

or do nothing --> does not leave starting bin/bowl/bag

YOUR GOAL

We will show you a **goal state**. That is, an example of how your pom poms should be sorted after your program is executed.

Your job is to write a *"program" using boolean expressions and conditionals* that explains how to properly sort the pompoms.

LET’S DO ONE TOGETHER!

BEFORE WE GO OUTSIDE

DECIDE ON YOUR GROUPS

Form teams of 4. You will then be paired with another team of 4, who will attempt to execute the code that you wrote (and vice versa).

THE RULES

Write your first program, switch places with your paired team and they attempt to execute your code 🤖 (and vice versa). On a success you move on to your second program. First pair of teams to complete all 4 programs wins.

REMINDERS

Remember the pompoms can be sorted into a bowl, a bag, or nothing (stays in the starting bin).

Also remember that each pompom has a color, a size, and is sparkly or not sparkly.

THE SOLUTIONS:

```
if (single_descriptor):
    put into bag OR bowl
elif (single_descriptor):
    put into bag OR bowl
else:
    put into bag OR bowl
```

```
if (descriptor AND descriptor):
    put into bag OR bowl
elif (descriptor OR descriptor):
    put into bag OR bowl
elif (NOT descriptor):
    put into bag OR bowl
```