# CONTROL FLOW

## Introduction to Control Flow

Imagine waking up in the morning.

You wake up and think, “Ugh. Is it a weekday?”

If so, you have to get up and get dressed and get ready for work or school. If not, you can sleep in a bit longer and catch a couple extra Z’s. But alas, it is a weekday, so you are up and dressed and you go to look outside, “What’s the weather like? Do I need an umbrella?”

These questions and decisions control the flow of your morning, each step and result is a product of the conditions of the day and your surroundings. Your computer, just like you, goes through a similar flow every time it executes code. A program will run (wake up) and start moving through its checklists, is this condition met, is that condition met, okay let’s execute this code and return that value.

This is the *control flow* of your program. In Python, your script will execute from the top down, until there is nothing left to run. It is your job to include gateways, known as [conditional statements](https://www.codecademy.com/resources/docs/python/conditionals?page_ref=catalog), to tell the computer when it should execute certain blocks of code. If these conditions are met, then run this function.

Diagram

Description automatically generated

Over the course of this lesson, you will learn how to build conditional statements using boolean expressions, and manage the control flow in your code.

## Conditionals

Conditionals take an expression, which is code that evaluates to determine a value, and checks if it is True or False. If it’s True, we can tell our program to do one thing — we can even account for False to do another.

As we write more complex programs, conditionals allow us to address multiple scenarios and make our programs more robust.

### If Statement

The Python if statement is used to determine the execution of code based on the evaluation of a Boolean expression.

* If the if statement expression evaluates to True, then the indented code following the statement is executed.
* If the expression evaluates to False then the indented code following the if statement is skipped and the program executes the next line of code which is indented at the same level as the if statement.

test\_value = 100

if test\_value > 1:

print("This code is executed!")

if test\_value > 1000:

print("This code is NOT executed!")

print("Program continues at this point.")

### Else Statement

The Python else statement provides alternate code to execute if the expression in an if statement evaluates to False.

The indented code for the if statement is executed if the expression evaluates to True. The indented code immediately following the else is executed if and only if the expression evaluates to False.

To mark the end of the else block, the code must be unindented to the same level as the starting if line.

test\_grade = 61

if test\_grade > 60:

print("You passed.")

else:

print("You failed.")

# Output: You passed.

### Elif Statement

The Python elif statement allows for continued checks to be performed after an initial if statement. An elif statement differs from the else statement because another expression is provided to be checked, just as with the initial if statement.

If the expression is True, the indented code following the elif is executed. If the expression evaluates to False, the code can continue to an optional else statement.

Multiple elif statements can be used following an initial if to perform a series of checks. Once an elif expression evaluates to True, no further elif statements are executed.

pet\_type = "fish"

if pet\_type == "dog":

print("You have a dog.")

elif pet\_type == "cat":

print("You have a cat.")

elif pet\_type == "fish":

# This is performed

print("You have a fish")

else:

print("Not sure!")

## Boolean Expressions

In order to build control flow into our program, we want to be able to check if something is true or not. A boolean expression is a statement that can either be True or False.

Let’s go back to the ‘waking up’ example. The first question, “Is today a weekday?” can be written as a boolean expression:

Today is a weekday.

This expression can be True if today is Tuesday, or it can be False if today is Saturday. There are no other options.

Consider the phrase:

Friday is the best day of the week.

Is this a boolean expression?

No, this statement is an opinion and is not objectively True or False. Someone else might say that “Wednesday is the best weekday,” and their statement would be no less True or False than the one above since there is no objective answer to what the best day of the week is.

How about the phrase:

Sunday starts with the letter 'C'.

Is this a boolean expression?

Yes! This expression can only be True or False, which makes it a boolean expression. Even though the statement itself is false (Sunday starts with the letter ‘C’), it is still a boolean expression.

### Exercise 1:

Determine if the following statements are boolean expressions or not. If they are, set the matching variable to the right to "Yes" and if not set the variable to "No". Here’s an example of what to do:

Example statement:

My dog is the cutest dog in the world.

This is an opinion and not a boolean expression, so you would set example\_statement to "No" in the editor to the right. Okay, now it’s your turn:

Statement one:

Dogs are mammals.

Statement two:

My dog is named Pavel.

Statement three:

Dogs make the best pets.

Statement four:

Cats are female dogs.

### Answer:

example\_statement = "No"

statement\_one = "Yes"

statement\_two = "Yes"

statement\_three = "No"

statement\_four = "Yes"

## Relational Operators: Equals and Not Equals

Now that we understand what boolean expressions are, let’s learn to create them in Python. We can create a boolean expression by using *relational operators*.

Relational operators compare two items and return either True or False. For this reason, you will sometimes hear them called [*comparators*](https://www.codecademy.com/resources/docs/python/operators?page_req=catalog).

The two relational operators we’ll cover first are:

* Equals: ==
* Not equals: !=

These operators compare two items and return True or False if they are equal or not.

We can create boolean expressions by comparing two values using these operators:

1 == 1     # True  
   
2 != 4     # True  
   
3 == 5     # False  
   
'7' == 7   # False

Each of these is an example of a boolean expression.

Why is the last statement false? The '' marks in '7' make it a string, which is different from the integer value 7, so they are not equal. When using relational operators it is important to always be mindful of type.

## Boolean Variables

Before we go any further, let’s talk a little bit about True and False. You may notice that when you type them in the code editor (with uppercase T and F), they appear in a different color than variables or strings. This is because True and False are their own special type: [bool](https://www.codecademy.com/resources/docs/python/data-types?page_ref=catalog).

True and False are the only bool types, and any variable that is assigned one of these values is called a *boolean variable*.

Boolean variables can be created in several ways. The easiest way is to simply assign True or False to a variable:

set\_to\_true = True  
set\_to\_false = False

You can also set a variable equal to a boolean expression.

bool\_one = 5 != 7   
bool\_two = 1 + 1 != 2  
bool\_three = 3 \* 3 == 9

These variables now contain boolean values, so when you reference them they will only return the True or False values of the expression they were assigned.

print(bool\_one)    # True  
   
print(bool\_two)    # False  
   
print(bool\_three)  # True

## If Statement

“Okay okay okay, boolean variables, boolean expressions, blah blah blah, I thought I was learning how to build control flow into my code!”

You are, I promise you!

Understanding boolean variables and expressions is essential because they are the building blocks of [*conditional statements*](https://www.codecademy.com/resources/docs/python/conditionals?page_ref=catalog).

Recall the waking-up example from the beginning of this lesson. The decision-making process of “Is it raining? If so, bring an umbrella” is a conditional statement.

Here it is phrased in a different way:

If it is raining, then bring an umbrella.

Can you pick out the boolean expression here?

Right, "it is raining" is the boolean expression, and this conditional statement is checking to see if it is True.

If "it is raining" == True then the rest of the conditional statement will be executed and you will bring an umbrella.

This is the form of a conditional statement:

If [it is raining], then [bring an umbrella]

In Python, it looks very similar:

if is\_raining:  
  print("bring an umbrella")

You’ll notice that instead of “then” we have a colon, :. That tells the computer that what’s coming next is what should be executed if the condition is met.

Let’s take a look at another conditional statement:

if 2 == 4 - 2:   
  print("apple")

Will this code print apple to the terminal?

Yes, because the condition of the if statement, 2 == 4 - 2 is True.

## Relational Operators II

Now that we’ve added conditional statements to our toolkit for building control flow, let’s explore more ways to create boolean expressions. So far we know two relational [operators](https://www.codecademy.com/resources/docs/python/operators?page_req=catalog), equals and not equals, but there are a ton (well, four) more:

* > greater than
* >= greater than or equal to
* < less than
* <= less than or equal to

Let’s say we’re running a movie streaming platform and we want to write a program that checks if our users are over 13 when showing them a PG-13 movie. We could write something like:

if age <= 13:  
  print("Sorry, parental control required")

This function will take the user’s age and compare it to the number 13. If age is less than or equal to 13, it will print out a message.

## Boolean Operators: and

Often, the conditions you want to check in your conditional statement will require more than one boolean expression to cover. In these cases, you can build larger boolean expressions using *boolean operators*. These operators (also known as *logical operators*) combine smaller boolean expressions into larger boolean expressions.

There are three boolean operators that we will cover:

* and
* or
* not

Let’s start with and.

and combines two boolean expressions and evaluates as True if both its components are True, but False otherwise.

Consider the example:

Oranges are a fruit and carrots are a vegetable.

This boolean expression is comprised of two smaller expressions, oranges are a fruit and carrots are a vegetable, both of which are True and connected by the boolean operator and, so the entire expression is True.

Let’s look at an example of some AND statements in Python:

(1 + 1 == 2) and (2 + 2 == 4)   # True  
   
(1 > 9) and (5 != 6)            # False  
   
(1 + 1 == 2) and (2 < 1)        # False  
   
(0 == 10) and (1 + 1 == 1)      # False

Notice that in the second and third examples, even though part of the expression is True, the entire expression as a whole is False because the other statement is False. The fourth statement is also False because both components are False.

## Boolean Operators: or

The boolean operator or combines two expressions into a larger expression that is True if either component is True.

Consider the statement

Oranges are a fruit or apples are a vegetable.

This statement is composed of two expressions: oranges are a fruit which is True and apples are a vegetable which is False. Because the two expressions are connected by the or operator, the entire statement is True. Only one component needs to be True for an or statement to be True.

In English, or implies that if one component is True, then the other component must be False. This is not true in Python. If an or statement has two True components, it is also True.

Let’s take a look at a couple of examples in Python:

True or (3 + 4 == 7)    # True  
(1 - 1 == 0) or False   # True  
(2 < 0) or True         # True  
(3 == 8) or (3 > 4)     # False

Notice that each or statement that has at least one True component is True, but the final statement has two False components, so it is False.

## Else Statements

As you can tell from your work with *Calvin Coolidge’s Cool College*, once you start including lots of if statements in a function the code becomes a little cluttered and clunky. Luckily, there are other tools we can use to build control flow.

else statements allow us to elegantly describe what we want our code to do when certain conditions are **not** met.

else statements always appear in conjunction with if statements. Consider our waking-up example to see how this works:

if weekday:  
  print("wake up at 6:30")  
else:  
  print("sleep in")

In this way, we can build if statements that execute different code if conditions are or are not met. This prevents us from needing to write if statements for each possible condition, we can instead write a blanket else statement for all the times the condition is not met.

Let’s return to our if statement for our movie streaming platform. Previously, all it did was check if the user’s age was over 13 and if so, print out a message. We can use an else statement to return a message in the event the user is too young to watch the movie.

if age >= 13:  
  print("Access granted.")  
else:  
  print("Sorry, you must be 13 or older to watch this movie.")

## Else If Statements

We have if statements, we have else statements, we can also have elif statements.

Now you may be asking yourself, what the heck is an elif statement? It’s exactly what it sounds like, “else if”. An elif statement checks another condition after the previous if statements conditions aren’t met.

We can use elif statements to control the order we want our program to check each of our conditional statements. First, the if statement is checked, then each elif statement is checked from top to bottom, then finally the else code is executed if none of the previous conditions have been met.

Let’s take a look at this in practice. The following if statement will display a “thank you” message after someone donates to a charity; there will be a curated message based on how much was donated.

print("Thank you for the donation!")  
   
if donation >= 1000:  
  print("You've achieved platinum status")  
elif donation >= 500:  
  print("You've achieved gold donor status")  
elif donation >= 100:  
  print("You've achieved silver donor status")  
else:  
  print("You've achieved bronze donor status")

Take a second to think about this function. What would happen if all of the elif statements were simply if statements? If you donated $1100.00, then the first three messages would all print because each if condition had been met.

But because we used elif statements, it checks each condition sequentially and only prints one message. If I donate $600.00, the code first checks if that is over 1000, which it is not, then it checks if it’s over 500, which it is, so it prints that message, then because all of the other statements are elif and else, none of them get checked and no more messages get printed.

# Review

Great job! We covered a ton of material in this lesson and we’ve increased the number of tools in our Python toolkit by several-fold. Let’s review what we’ve learned this lesson:

* Boolean expressions are statements that can be either True or False
* A boolean variable is a variable that is set to either True or False.
* We can create boolean expressions using relational operators:
  + ==: Equals
  + !=: Not equals
  + >: Greater than
  + >=: Greater than or equal to
  + <: Less than
  + <=: Less than or equal to
* if statements can be used to create control flow in your code.
* else statements can be used to execute code when the conditions of an if statement are not met.
* elif statements can be used to build additional checks into your if statements