

# **Control Flow: Conditionals**

## Introduction

We have talked about different data types, how to use them, and the kinds of operations that we can perform on them. We have also talked about using Booleans ( True or False ) to inform decisions in our programming. Often when we want to implement a decision in our code, we'll use conditionals. Conditionals allow us to break up our code in a way that we can selectively perform operations like assigning a value or even just printing text.

# **Objectives**

You will be able to:

Use Python conditional statements

#### **Execution Flow**

So far in Python, all of our lines of code run one after the other. So in the code below, vacation\_days is initially assigned to 0, then it is reassigned by incrementing by one, and again reassigned by incrementing again by one, which brings the vacation\_days to a total of 2.

```
# set-up vacation_days variable
vacation days = 0
# add 1 to vacation_days and rewrite itself
vacation days = vacation days + 1
vacation days = vacation days + 1
# print how many vacation days there are
print(vacation days)
2
# reset variable to 0
vacation_days = 0
# this does the same as above
vacation days += 1
vacation_days += 1
# print how many vacation days there are
print(vacation_days)
2
```

The += is used to increment the current value of the variable and assign this new value back to it. The statement vacation\_days += 1 can be thought of as vacation\_days = vacation\_days + 1. On line 1 we assign vacation\_days the value 0. So, on line two, we reassign vacation\_days to equal the current value of vacation\_days, which is 0, plus 1. Again we increment vacation\_days on line 3, which would now equate to 1 + 1, and finally we output the new value of vacation\_days, 2.

### If Statement

In Python there are three conditional statements, if, elif, and else. Every conditional statement is required to begin with an if. The elif and else statements are not always required. elif is short for else if.

Now what if we wanted to only increment our vacation days based on a condtion? We could imagine a condition in this context being whether or not we hit our goals this quarter at work. If that condition were True, we would want to increase our vacation days. However, if that condition is not True, we can't increase our vacation time. Let's look at the code below that contains an if statement.

Code that is part of an if statement *block* runs only when the condition evaluates to True. So if the condition evaluates to False our block of code will not be run and it moves on to the next block.

**Note:** A *block* is any code that is grouped together. With conditionals, we indicate that something is part of the *block* by *indentation*. So the line vacation\_days += 1 is indented to ensure that it is run as a part of the conditional argument below. To end the block we simply stop indenting.

**Note:** In all of the following code blocks, pay close attention to how the variable vacation\_days is affected depending on the value and order of the conditions. The idea is to understand how Python decides which code should run when it comes across an if, elif, and else statement.

```
vacation_days = 1
goals_met = True
if goals_met:
    # all code indented under the if statement is the block
    # indented code runs since conditional argument is True
    vacation_days += 1
    print("vacation_days = ", vacation_days)
    print("we incremented vacation days")
# if block ends

vacation_days = 2
we incremented vacation days
```

Because our condition was True, our block of code ran. We added 1 day to our vacation days variable and printed two statements.

What if the first conditional was set to False but we wanted to make sure there is *some* code that runs no matter what? That is where we can use our else statement.

```
vacation_days = 1
goals met = False
if goals_met:
    # if block starts
    # code does not run since conditional argument is False
    vacation days += 1
    print("vacation_days = ", vacation_days)
    print("we incremented vacation days")
# if block ends
else:
    # else block starts
    print("vacation_days = ", vacation_days)
    print("we did NOT increment vacation days")
# else block ends
vacation_days = 1
we did NOT increment vacation days
```

Above we can see that the condition following the if is False and the code directly underneath is not run. The variable vacation\_days stays assigned to the number 1. However, since we now have an else statement, our if block gets skipped and we then move on to the block of code underneath our else statement, which we can see prints the number of vacation days and a message indicating that we did not increment the vacation days.

But what if we have a second condition? Let's say we really worked hard and exceeded all our goals for the quarter? Maybe instead of incrementing just 1 day, we increment our vacation days by 2! The third conditional statement is an elif statement, which is essentially another if statement that follows the first if statement. elif still requires a condition and a block of code, but it only comes after an if statement or another elif statement.

Let's take a look:

```
vacation_days = 1
goals_met = False
goals_exceeded = True
if goals_met:
    # code does not run since conditional argument is False
    vacation_days += 1
    print("vacation_days = ", vacation_days)
    print("we incremented vacation days")
elif goals_exceeded:
```

```
print("We are now in our elif statement!")
  print("This means that we exceeded our goals this quarter")
  print("We will increase our vacation days by two")
  vacation_days += 2
  print("vacation_days = ", vacation_days)

else:
  print("vacation_days = ", vacation_days)
  print("we did NOT increment vacation days")

We are now in our elif statement!
This means that we exceeded our goals this quarter
We will increase our vacation days by two
vacation_days = 3
```

It is important to note that an else block comes last and will **only** run if all the conditions before it are false.

### **Truthiness**



So far our conditionals have depended on whether something evaluates exactly to True or False. But conditionals don't force us to be so precise. Conditionals also consider some values True if they are truthy and False if they are falsy. Take a look at the following:

```
vacation_days = 1
if vacation_days:
    # this is run
    vacation_days += 1
vacation_days
```

Even though vacation\_days did not evaluate to True, it still ran the code in the if block because the value for vacation\_days was 1, which is considered truthy.

So, from this we can surmise that numbers are **truthy** values. EXCEPT, in Python 0 is **not** considered a truthy value.

```
vacation_days = 0
if vacation_days:
    # this is not run
    vacation_days += 1
vacation_days
```

If 0 is **not truthy**, and it is not True or False, what is it? Just as Python has **truthy** values, it has **falsy** values (or values that are treated as False in conditional statements), and 0 is considered a falsy value.

Above, we can see that the if block was not run and vacation\_days was not incremented, almost as if vacation\_days evaluated to False.

So what is truthy and what is falsy in Python? Zero is falsy, and None is falsy. Also falsy are values like empty strings ("") and empty lists ([]), which we will learn more about in later lessons. Let's take a look at this.

```
greeting = ''
if greeting:
    greeting += 'Hello'
else:
    greeting += 'Goodbye'
greeting
'Goodbye'
```

If we are ever curious about the whether something is truthy or falsy in Python, we can just ask with the bool function.

```
bool(0) # False

False

bool(1) # True
```

```
True
bool('')
False
bool([])
```

Boolean values (True and False) can also be used in mathematical equations. True is set to 1 and False is set to 0.

```
True + 5 + True

7

True - False - False + True

2

True * 4
```

We can see how this can be put to use in an If statement

```
test_var = 1
if test_var:
    print('Truthy')
```

Truthy

## **Using Data to Make Decisions**

Our code in conditional arguments becomes more interesting when we use conditional arguments that are less direct than just True or False.

Let's say you have a management system that allows employees to put in their vacation days and the system has a way of auto tagging vacations as disruptive, average, or minimal based on the number of days being taken off. This helps to give immediate feedback for you so that you can make sure to plan deadlines and staff projects appropriately. So, we'll use conditional statements that use the number of days taken off as the input to figure out if the vacation is disruptive, normal, or minimal.

Let's start out with just creating logic that creates a tag for disruptive vacations or vacations longer than 5 days off of work. All other vacations can be tagged as average.

```
number_of_days = 7
if number_of_days > 5:
    print("#disruptive")
else:
    print("#average")
#disruptive
```

Great, now what if we want to tag vacations that are under 3 days off as minimally disruptive? Well, we don't want to create a new if block entirely. If we do, then our else statement will always run and we'll get #average and #minimal, which is not what we want.

So, we'll have to use an elif statement.

```
number_of_days = 2
if number_of_days > 5:
    print("#disruptive")
elif number_of_days < 3:</pre>
```

```
print("#minimal")
else:
    print("#average")

#minimal

number_of_days = 3
if number_of_days > 5:
    print("#disruptive")
elif number_of_days < 3:
    print("#minimal")
else:
    print("#average")</pre>
```

We can see how powerful this kind of code can be to creating dynamic and efficient programs and to makings decisions. Now, there is no need for anyone to manually rank each employee's vacations and we now have a system that flags vacations that might need some special planning by management in order to keep work running smoothly.

# **Summary**

In this lesson, we saw how conditionals allow us to make decisions with our code by only executing code under the <code>if</code> statement when the conditional argument is <code>True</code> or truthy. We then saw how we can use the <code>else</code> statement to only run code when the conditional argument is <code>False</code> or falsy, and as we know, code that is not in a conditional block is still run as normal. Next, we examined what is truthy or falsy, and saw that None, 0, empty strings, and lists are all falsy. If we are unsure, we can use the <code>bool</code> function to see the boolean equivalent of a piece of data. Finally, we used <code>if</code>, <code>elif</code>, and <code>else</code> statements together to make decisions based on the conditions of our problem.

#### Releases

No releases published

#### **Packages**

#### Contributors 7













#### Languages

Jupyter Notebook 100.0%