3.2.8 Decision Statements

Now that you are familiar with data storage and retrieval, Seth and Tom would like you to learn how to retrieve data based on whether a condition is true or false. Retrieving data based on a condition is known as a decision statement. Decision statements are widely used by programmers because they often need to retrieve data from a large dataset that meets certain criteria. Tom is going to walk you through the structure of a simple decision statement and demonstrate how to create and use them. Then, he'll show you how to create and implement more complex decision statements.

Some programming problems can be solved by performing a series of ordered steps. For example, if you need to create an algorithm that calculates the percentage of votes a candidate receives in an election, you might write a simple algorithm like this:

```
# How many votes did you get?
my_votes = int(input("How many votes did you get in the election? "))
#  Total votes in the election
total_votes = int(input("What is the total votes in the election? "))
# Calculate the percentage of votes you received.
percentage_votes = (my_votes / total_votes) * 100
print("I received " + str(percentage_votes)+"% of the total votes.")
```

The steps represented in this algorithm are:

1. Declare the "my_votes" variable equal to an `input` function. The `input` function will prompt the user to type an amount, such as 200.

2. Wrap the `input` function with the `int()` method. When we use an `input` function, the data type of the user input defaults to a string. The `int()` method converts the user input value to an integer, which is necessary to perform the calculation for the percentage of votes. If we want the user to enter a floating-point decimal number, then we would use the `float()` method.
3. Calculate the percentage of votes by dividing the users' votes by the total votes, and then multiplying by 100 using the multiplication operator, the asterisk.
4. Create a print statement that tells the percentage of votes. The `percentage_votes` must be converted from `int` to a string data type using `str()`, which is necessary in order to print the percentage of votes in the sentence.

NOTE

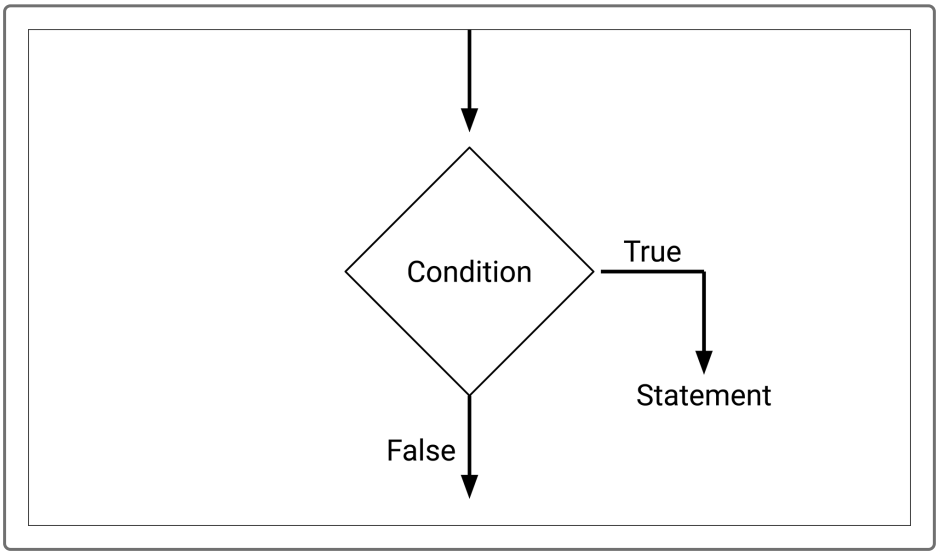
> The `input` function asks a user for text input. Once the user inputs text and presses Enter, the program will resume.

However, if you needed to create an algorithm that determines if a candidate won the election based on the majority of the votes, which is equal or greater than 50.1%, then the above program would require a control structure to execute a set of statements only under certain conditions. This type of program is called a **decision statement** or decision structure.

We'll cover two types of decision statements: `if` statements and `if-else` statements.

If Statements

In a simple decision structure like an `if` statement, if a condition is met (i.e., is true), then an action is performed. If the condition is not met (i.e., is false), then the action is not performed. This can be illustrated with the following flowchart:



In Python, the general format for the `if` statement is to write a single alternative decision as follows:

```
if condition:
   statement 1
   statement 2
```

REWIND

Remember, `if` statements tell the computer that certain lines of code should only run under certain conditions. The `if` statement checks if a condition is true. If it's true, then a block of code below it will run.

Here are the general rules when using `if` statements in Python:

- `If` statements begin with the word "if," followed by a condition. The condition can be an expression like `if len(counties) > 2:`, which will be evaluated as either true or false. At the end of the condition is a colon.
- The next line contains a block of statements. All statements in a block must be consistently indented. This indentation is required because the Python interpreter or VS Code editor uses it to determine

where the block begins and ends. If you do not indent, your code will not be executed.

- If the statement is not met (i.e., is false), then the statements in the block are skipped.

Let's practice using the `if` statement on our counties list to determine if the second county in the list is Denver. If so, then we will print "Denver" to the screen.

Start by opening VS Code and creating a new file called `Python_practice.py` in the Election_Analysis folder.

REWIND

Add the `.py` extension to the Python file so that you can run it.

Then, add the following code to the file, save the file, and then run the file in the VS Code terminal.

```
counties = ["Arapahoe","Denver","Jefferson"]
if counties[1] == 'Denver':
    print(counties[1])
```

This code might appear straightforward, but let's break down what's happening here.

- In the second line, we create our if statement and provide the condition, `counties[1] == 'Denver'`.
- The double equals sign, `==`, is a comparison operator, or Boolean operator, which means "equal to."
- Since the second item in the counties list, `counties[1]`, is "equal to" `'Denver'`, the condition is met. "Denver" is printed to the terminal screen.

REWIND

In Python, we can access the items in a list using an index.

Comparison Operators

Comparison operators, or **Boolean operators,** are used to compare values, which will return `True` or `False` according to the condition. The following table highlights the different comparison or Boolean operators and their meanings.

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

Retake

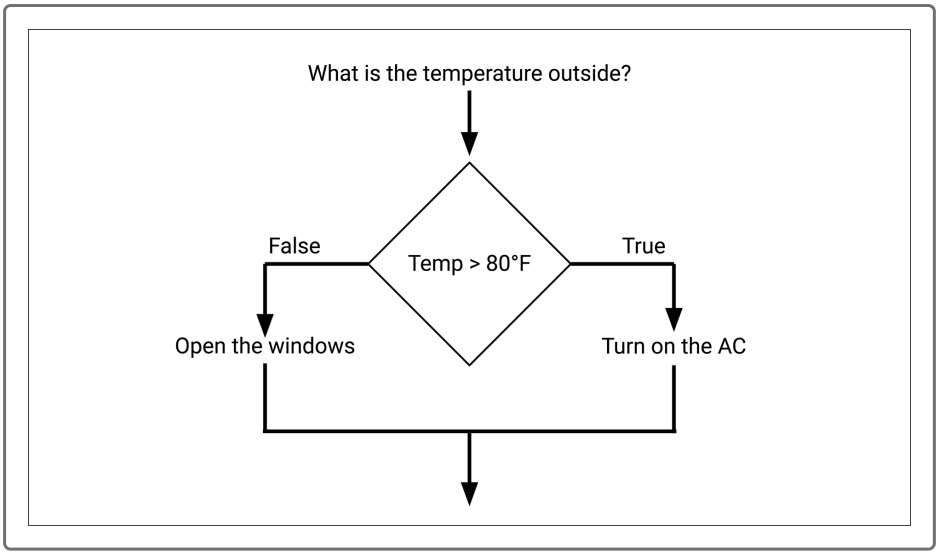
NOTE

> For more information, see the **documentation on comparison operators in Python (https://docs.python.org/3.7/library/stdtypes.html#comparisons)** .

If-Else Statements

The `if-else` statement is used when we need an outcome for both true and false conditions. The `if-else` statement is also referred to as a **dual-alternative decision statement**.

An `if-else` statement will execute one block of statements, or path, if a condition is true, or another block of statements if the condition is false. This can be illustrated with the following flowchart.



In Python, the general format for the `if-else` statement is to write a dual-alternative decision as follows:

```
if condition:
   statement 1
   statement 2
else:
   statement 1
   statement 2
```

Here's how an `if-else` statement works.

- The `if-else` statement begins with the word "if," followed by a condition.
- The condition is tested.
- If the condition is true, the block of indented statements following the `if` statement is executed, and the program moves out of the `if-else` statement.

- If the condition is false, the program skips to the `else` statement, the block of indented statements following the `else` statement is executed, and the program moves out of the `if-else` statement.

IMPORTANT

> When writing an `if-else` statement, follow these general rules for indentation:
>
> - Make sure the `if`statement and the `else` statement are aligned.
> - Make sure the statements under the `if` and `else` statements are consistently indented.

To demonstrate how an `if-else` statement works, let's write a Python script that matches our flowcharts. Create a new Python file and add the following code. Then run the file in the VS Code terminal.

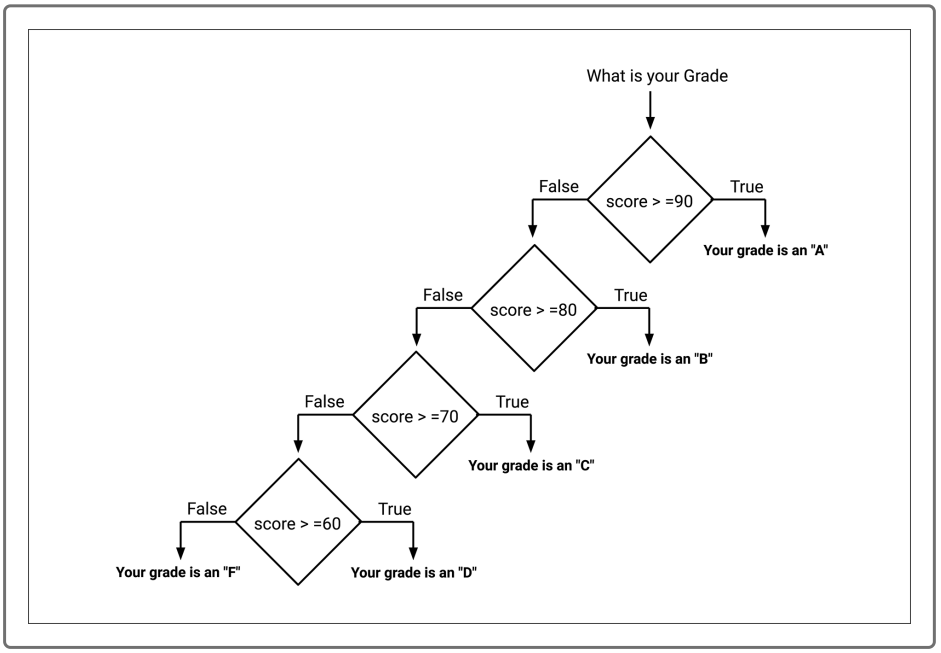
```
temperature = int(input("What is the temperature outside? "))

if temperature > 80:
    print("Turn on the AC.")
else:
    print("Open the windows.")
```

Here, the temperature variable is declared by asking the user to enter the outside temperature with an `input` statement wrapped in the `int()` method. The `int()` method will convert the user input data type from a string to an integer. The integer is used to assess the `if-else` statement.

Nested If-Else Statements

Sometimes, a decision structure can be more complex than a dual-alternative decision structure. For instance, it's not uncommon for decision structures to be nested inside another decision structure. An example of this is writing an algorithm to determine a letter grade based on a number. Take a look at the following chart.



The general rules for the nested `if-else` decision structure example above are the same as for a single dual `if-else` statement.

To demonstrate how a nested `if-else` statement works, let's write a Python script that matches the flowchart. Create a new Python file and add the following code. Then run the file in the VS Code terminal.

```
#What is the score?
score = int(input("What is your test score? "))

# Determine the grade.
if score >= 90:
    print('Your grade is an A.')
else:
    if score >= 80:
        print('Your grade is a B.')
    else:
        if score >= 70:
            print('Your grade is a C.')
        else:
            if score >= 60:
                print('Your grade is a D.')
            else:
                print('Your grade is an F.')
```

These nested `if-else` statement is quite complex and has a special name: `if-elif-else` statements, or the `if-elif-else` statement.

With nested `if-elif-else` statements, we can create a compound statement, `elif`, in the following manner.

- The first `if` statement tests the condition.
- The following `else` statement and the preceding `if` statement are compounded to make the `elif`statement.
- This syntax continues until the last `else` statement.
- In an `if-elif-else` statement, the `if, elif`, and `else` statements are all aligned, and the conditionally executed blocks are indented.

Using these rules for the `if-elif-else` statement, let's rewrite the code and run the file in the VS Code terminal to determine a letter grade.

```
# What is the score?
score = int(input("What is your test score? "))

# Determine the grade.
if score >= 90:
    print('Your grade is an A.')
elif score >= 80:
    print('Your grade is a B.')
elif score >= 70:
    print('Your grade is a C.')
elif score >= 60:
    print('Your grade is a D.')
else:
    print('Your grade is an F.')
```

Compared to the nested `if-elif` statement, the `if-elif-else` statement tends to be easier to read. This is because the `if`, `elif`, and `else` statements are aligned, and the block statements are usually shorter.

SHOW HINT