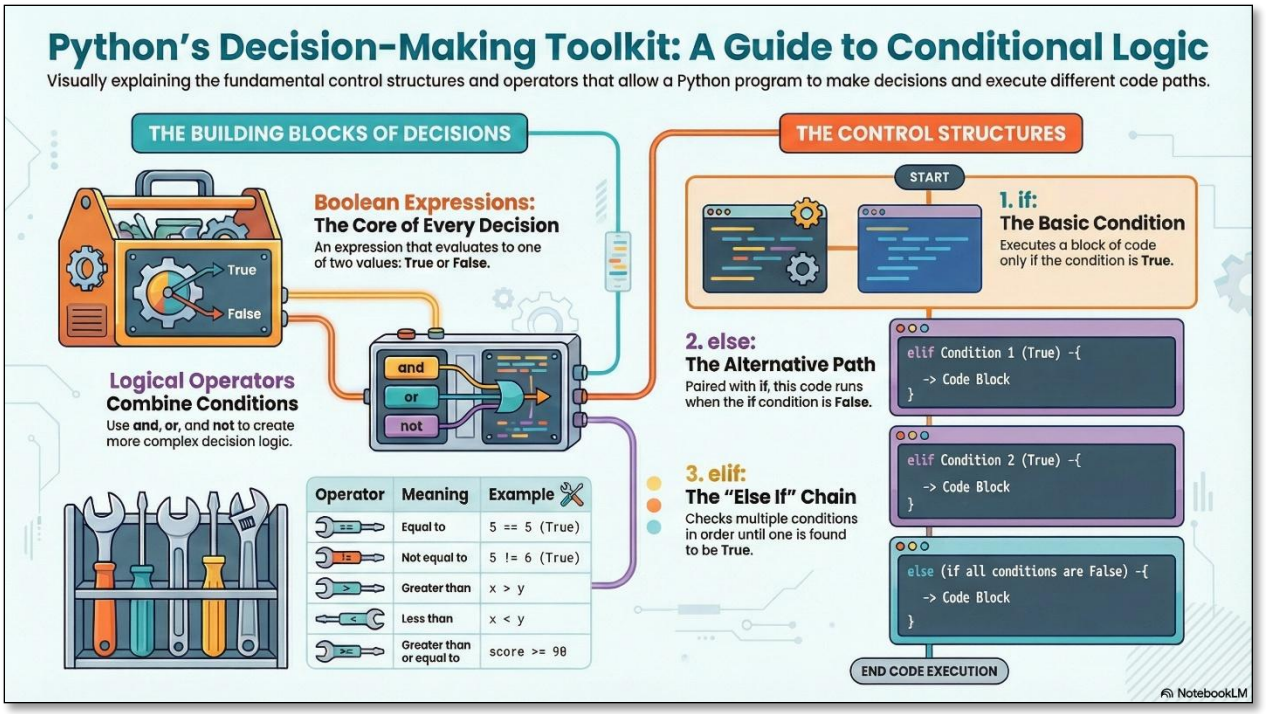
Chapter 3: Decisions, Decisions, Decisions

Contents



Red light: No AI

Time required: 180 minutes

NOTE: Please complete all tutorials and assignments.

Online Tutorials

You may want to go through these short online tutorials to supplement your learning.

- Python Booleans
- Python If … Else
- Python Introduction
- Python Get Started
- Python Syntax
- Python Comments
- Python Variables
    - Variable Names

    - Assign Multiple Values
    - Output Variables
    - Global Variables
- Python Data Types
- Python Numbers
- Python Casting
- Python Strings
- Python Operators
- Python User Input

DRY

Don't **R**epeat **Y**ourself

Visualize and Debug Programs

The website www.pythontutor.com helps you create visualizations for the code in all the listings and step through those programs one instruction at a time. As you do that, pay attention to the movements of the red and green arrows on the left, the diagram on the right, and the printed material at the bottom. This will help you to better understand the behavior of the programs we are working on.

This is a great way to debug your code. You can see the variables change as you step through the program.

www.pythontutor.com

Objectives

In this chapter you will learn about:

- The if Statement
- Logical Operators
- Boolean Expressions and Variables
- Conditional Execution

- String Comparison
- Alternative Execution - if else
- Chained Conditionals – if elif else
- Nested Conditionals
- Short-Circuit Evaluation of Logical Expressions

Control Structures

A control structure in Python helps manage the flow of code execution. There are three main types.

1. **Sequential:** Code runs line by line, executing one after the other.
2. **Selection (Conditional): if**, **elif**, and **else** statements allow different paths based on conditions.
3. **Iteration (Loops): for** and **while** loops help execute code repeatedly until a condition is met.

Everything we have done to this point has been sequential.

A control structure like the **if** statement allows a program to check conditions and change the behavior of the program. This allows the computer to make decisions based on input.

This decision is based on whether the result of the if statement is true or false. This is called a Boolean expression.

Boolean Expression

A Boolean expression is an expression that is either true or false. True and False are called "Boolean values" that are predefined in Python. True and False are the only Boolean values, and anything that is not False, is True.

The following expressions use a **comparison operator ==**, which compares two operands and produces **True** if they are equal and **False** otherwise:

```
5 == 5
True
5 == 6
False
```

True and False are special values that belong to the class bool; they are not strings:

```
type(True)
<class 'bool'>
type(False)
<class 'bool'>
```

The == operator is one of the comparison operators; the others are:

x == y	x is equal to y
x != y	x is not equal to y
x > y	x is greater than y
x < y	x is less than y
x >= y	x is greater than or equal to y
x <= y	x is less than or equal to y
x is y	x is the same as y
x is not y	x is not the same as y

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols for the same operations. A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a comparison operator. There is no such thing as =< or =>.

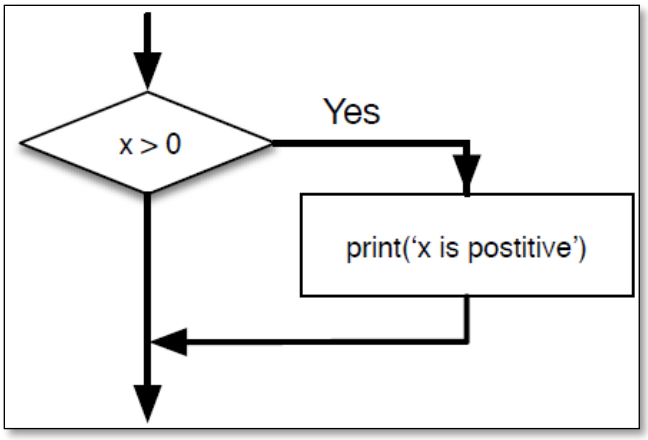
Tutorial 3.1: The if Statement

Conditional execution in Python utilize **if**, **elif** (optional), and **else** statements to execute code based on conditions.

Indentation is crucial; Python uses it to define code blocks.

The expression must be one that will evaluate to either **True** or **False**.

1. If the expression evaluates to **True**, the indented statement(s) that follow the colon will be executed in sequence.
2. If the expression evaluates to **False**, those indented statements will be skipped and the next statement following the if statement will be executed.



Create a Python program named **if.py**

```
x = 1
# Is x positive or greater than 0?
if x > 0:
    print('x is positive')
```

Example run:

```
x is positive
```

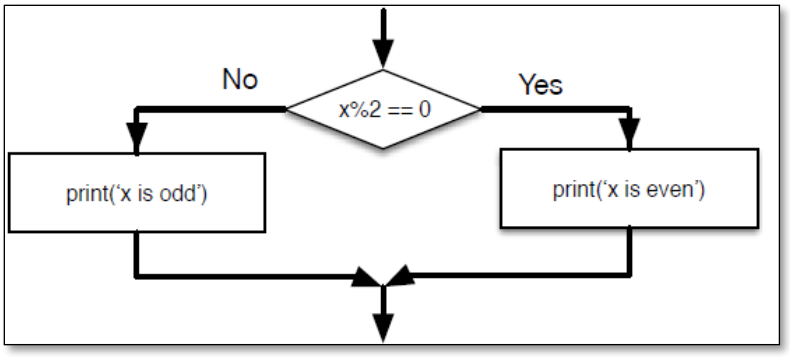
The Boolean expression after the **if** statement is called the **condition**. The **if** statement ends with a colon character (:) and the line(s) after the **if** statement are indented.

If the logical condition is **True**, then the indented statement gets executed. If the logical condition is **False**, the indented statement is skipped.

Tutorial 3.2: The else Statement

The **else** statement is used with the **if** statement. It provides a way to specify a block of code that should be executed when the condition of the **if** statement evaluates to **False**.

If the remainder when x is divided by 2 is 0, we know that x is even, and the program displays a message to that effect. If the condition is **False**, the second set of statements is executed.



Since the condition must either be true or false, exactly one of the alternatives will be executed. The alternatives are called branches because they are branches in the flow of execution.

The **else** statement is optional and helps handle situations when the **if** condition is not met. It's a way to define an alternative action when the condition isn't satisfied.

Create a Python program named: **odd_or_even.py**

```
1   x = int(input("Enter a number: "))
2
3   # % is the modulo operator
4   # This returns the remainder of integer division
5   # If the remainder is 0, then the number is even
6   if x % 2 == 0:
7       print("x is even")
8   else:
9       print("x is odd")
```

Example runs:

```
Enter a number: 7
x is odd
PS Z:\_WNCC\Pythor
Enter a number: 8
x is even
```

String Comparison

Comparison operators work for strings.

```
truck = 'Dodge'
if truck == 'dodge':
   print('You have a Dodge truck.')
else:
   print('You don’t have a Dodge truck.')
```

```
You don't have Dodge truck.
```

Python, like many programming languages, is case sensitive. Dodge is not the same as dodge. Use the **.lower()** operator to make sure you are comparing strings properly.

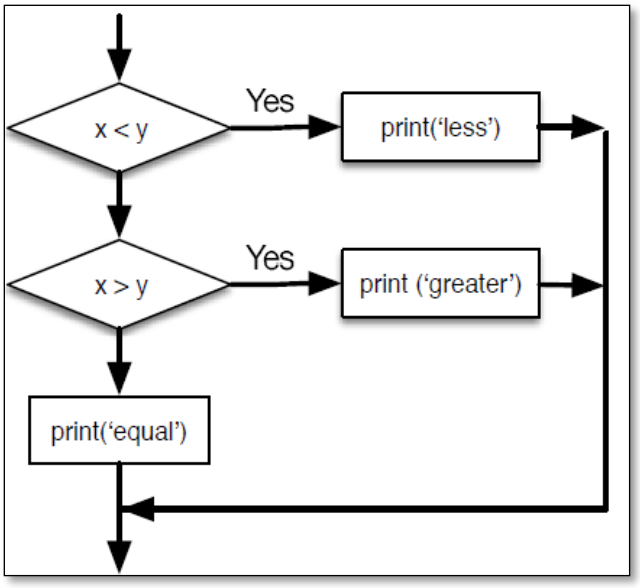
```
truck = 'DoDgE'
if truck.lower() == 'dodge':
   print('You have a Dodge truck.')
else:
   print('You don’t have a Dodge truck.')
```

```
You have a Dodge truck.
```

Tutorial 3.3: The elif Statement

Here are the three keywords that can be used with **if** statements:

- **if**
- **elif**
- **else**



If there are more than two possibilities, we need more than two branches. One way to express a computation like that is a chained conditional:

```
if x < y:
   print('x is less than y')
elif x > y:
   print('x is greater than y')
else:
   print('x and y are equal')
```

elif is an abbreviation of "else if." Exactly one branch will be executed. There is no limit on the number of **elif** statements. If there is an **else** clause, it has to be at the end, but there doesn't have to be one.

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

You can use an **if** statement to assign letter grades. Suppose that scores 90 and above are A's, scores in the 80s are B's, 70s are C's, 60s are D's, and anything below 60 is an F.

Create and save the following program as **grades.py**.

```python
"""
    Name: grades.py
    Author:
    Created:
    Purpose: Determine a letter grade from a score
"""

# Get score from the user
score = int(input("Enter your score: "))

# Determine what the grade is and display it
if score >= 90:
    print("A")
elif score >= 80:
    print("B")
elif score >= 70:
    print("C")
elif score > 60:
    print("D")
else:
    print("F")
```

Example run:

```
Enter your score: 87
B
```

Using **elif**, as soon as we find where the score matches, we stop checking conditions and skip all the way to the end of the whole block of statements.

You will rarely use more than one **if** statement in a multiple condition test. Use **elif** as shown above.

Assignment 3.1: Temperature Advisor

Objective:
Give weather advice based on temperature.

Instructions:

Create a Python program named: **temperature_advisor.py**

- Ask the user to input the current temperature in Fahrenheit.

- Use if, elif, and else to print the appropriate message.
    - 85 or more: "It's hot outside!"
    - 60–84: "Nice weather today."
    - 40–59: "It's a bit chilly."
    - Below 40: "Brrr... it's cold!"

Example run:

```
Enter the temperature in Fahrenheit: 87
It's hot outside!
```

Logical Operators

There are three logical operators: **and**, **or**, and **not**. The semantics (meaning) of these operators are like their meaning in English.

and	And	a and b	true if both a and b are true
or	Inclusive OR	a or b	true if either a or b are true
not	Negation	not a	switch a from true to false or from false to true

For example,

```
x > 0 and x < 10
```

is true only if x is greater than 0 and less than 10.

```
n % 2 == 0 or n % 3 == 0
```

is true if either of the conditions is true, that is, if the number is divisible by 2 or 3.

Finally, the **not** operator negates a Boolean expression, so not (x > y) is true if x > y is false; that is, if x is less than or equal to y.

```
print((9 > 7) and (2 < 4)) # Both original expressions are True
print((8 == 8) or (6 != 6)) # One original expression is True
print(not(3 <= 1)) # The original expression is False
```

Tutorial 3.5: Logical Operators

Create a Python program named **logical_operators.py**

```python
# Filename: logical_operators.py

# Getting user input for age
age = int(input("How old are you? "))

# Using 'and' logical operator to check if
# age is between 16 and 65 (inclusive)
if age >= 16 and age <= 65:
    print("Have a good day at work")
else:
    print("Enjoy your free time")

# Using 'or' logical operator to check if
# age is less than 16 or greater than 65
if age < 16 or age > 65:
    print("Enjoy your free time")
else:
    print("Have a good day at work")
```

Example run:

```
How old are you? 15
Enjoy your free time
Enjoy your free time
```

```
How old are you? 63
Have a good day at work
Have a good day at work
```

Notice that each conditional returns the same answer. These are two different ways to solve the same problem.

Nested Conditionals

One conditional can also be nested within another. We could have written the three-branch example like this:

```
if x == y:
   print("x and y are equal"
else:
   if x < y:
      print("x is less than y")
   else:
      print("x is greater than y")
```

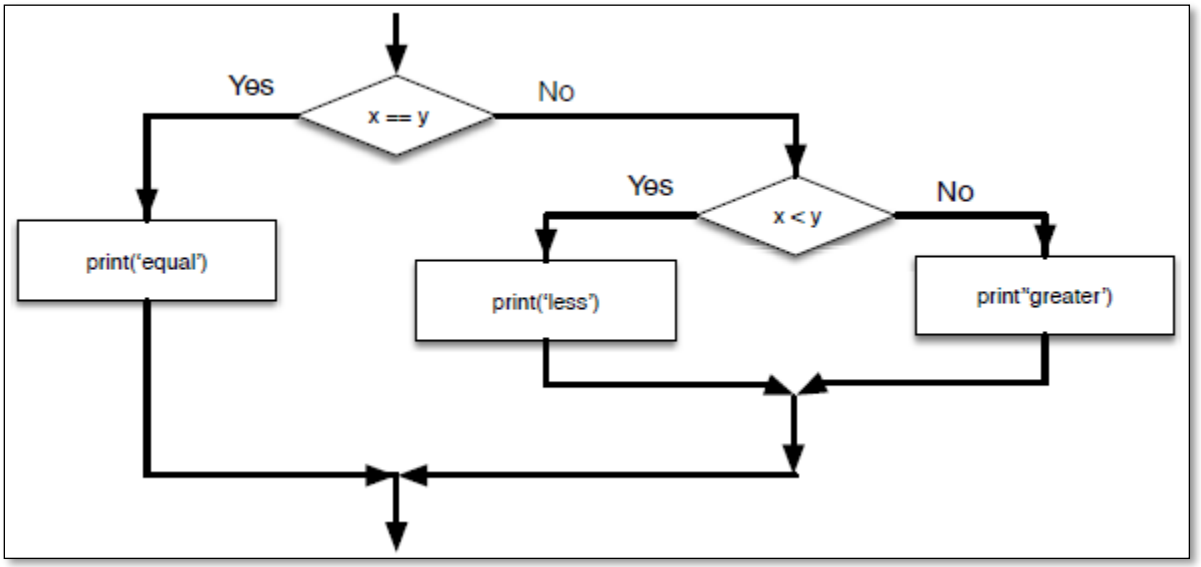
The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another if statement, which has two branches of its own. Those two branches are both simple statements, although they could have been conditional statements as well.

Although the indentation of the statements makes the structure apparent, nested conditionals become difficult to read very quickly. In general, it is a good idea to avoid them when you can.

Logical operators often provide a way to simplify nested conditional statements. For example, we can rewrite the following code using a single conditional:

```
if 0 < x:
   if x < 10:
      print("x is a positive single-digit number.")
```

The **print** statement is executed only if we make it past both conditionals, so we can get the same effect with the and operator:



```
if 0 < x and x < 10:
   print("x is a positive single-digit number. ")
```

Debugging

The traceback Python displays when an error occurs contains a lot of information, but it can be overwhelming. The most useful parts are usually:

- What kind of error it was
- Where it occurred

Syntax errors are usually easy to find, but there are a few gotchas. Whitespace errors can be tricky because spaces and tabs are invisible; we are used to ignoring them.

```
File "<stdin>", line 1
y = 6
^
IndentationError: unexpected indent
```

In this example, the problem is that the second line is indented by one space. But the error message points to y, which is misleading. In general, error messages indicate where the problem was discovered, but the actual error might be earlier in the code, sometimes on a previous line.

In general, error messages tell you where the problem was discovered, but that is often not where it was caused.

Assignment 2.2: Club Bouncer for Taylor Swift

Taylor Swift is coming to town. She is playing at the local school auditorium. You want to buy a ticket. Before you can do that, you must write a program to determine who can enter the concert.

When you are testing a single variable for a range of values, when you have eliminated the upper range, you don't have to test that any more.

For example: If someone is not older than 21, you don't have to test again for not being older than 21.

NOTE: You can't have more than one age. You would not use multiple if statements. We will rarely if ever use multiple if statements for a single variable.

1. Ask for the user's age.

2. If they are over 21 → they can enter and have a drink.
3. Else if they are between 18 and 21 inclusive (18, 19, 20, or 21) → They can enter but can't have a drink. They must wear a wristband.
4. Otherwise, they are younger than 18 → They are too young.

TODO

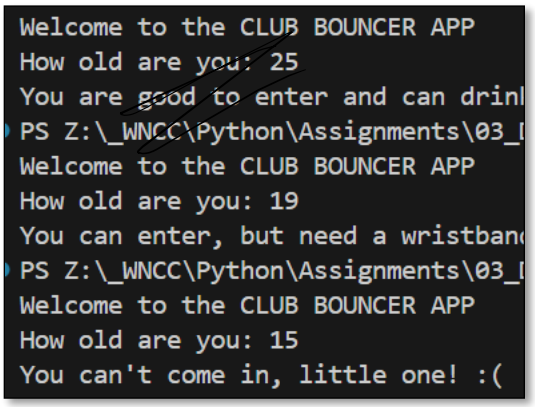
```
# Filename: bouncer.py

# TODO: Print a program title

# TODO: Get user input for age and convert it to an integer

# TODO: Check if the age meets the conditions for entry, print result
# 21+ drink, 18+ can enter, <18 can't enter
```

Example run:



Glossary

body The sequence of statements within a compound statement.

Boolean expression An expression whose value is either True or False.

branch One of the alternative sequences of statements in a conditional statement.

chained conditional A conditional statement with a series of alternative branches.

comparison operator One of the operators that compares its operands: ==, !=,

>, <, >=, and <=.

conditional statement A statement that controls the flow of execution depending on some condition.

condition The Boolean expression in a conditional statement that determines which branch is executed.

compound statement A statement that consists of a header and a body. The header ends with a colon (:). The body is indented relative to the header.

guardian pattern Where we construct a logical expression with additional comparisons to take advantage of the short-circuit behavior.

logical operator One of the operators that combines Boolean expressions: and, or, and not.

nested conditional A conditional statement that appears in one of the branches of another conditional statement.

traceback A list of the functions that are executing, printed when an exception occurs.

short circuit When Python is part-way through evaluating a logical expression and stops the evaluation because Python knows the final value for the expression without needing to evaluate the rest of the expression.

Assignment Submission

1. Attach all tutorials and assignments.
2. Attach screenshots showing the successful operation of each tutorial program.
3. Submit in Blackboard.