Module Learning Outcomes

After successful completion of this module, you will be able to ...

1. Use comparison and logical operators to write conditions.
2. Write if statements to create branch points in your code.
3. Write while loops and for loops to repeat sections of code.
4. Recall some techniques for tracing code.

Key questions:

* What does it mean to "nest" if statements or loops?
* What significance does indentation have in Python?

Explorations

Use the pages within this module to explore the following concepts:

* Exploration: [Comparison operators and logical operators](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-comparison-operators-and-logical-operators) (CLO 1b, MLO 1)
* Exploration: [Conditional execution](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-if-statements) (CLO 1a, MLO 2)
* Exploration: [Iteration (looping)](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-while-loops-and-for-loops) (CLO 1a, MLO 3)
* Exploration: [Code tracing](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-code-tracing) (CLO 2)
* [Animation and video demos](https://canvas.oregonstate.edu/courses/1928696/pages/module-3-animation-and-video-demos) (CLO 1a, 1b, MLO 1-3)
* [Module 3 Exercise Solutions](https://canvas.oregonstate.edu/courses/1928696/pages/module-3-exercise-solutions)

Optional Resources

* [*Think Python* Chapter 5 sections 1 - 6Links to an external site.](http://greenteapress.com/thinkpython2/html/thinkpython2006.html)
* [*Think Python* Chapter 7Links to an external site.](http://greenteapress.com/thinkpython2/html/thinkpython2008.html)

Task List

Complete the following assignments and other tasks:

* Read the Exploration pages and do the interactive exercises on those pages (CLO 1a-b, MLOs 1-3).
* Write your [Project plan for Project 3a](https://canvas.oregonstate.edu/courses/1928696/assignments/9073296) (CLOs 1a-b, 4, MLOs 1-3). Note that it's due a few days before project 3.
* Review the [Pseudocode and flowchart guidelines](https://canvas.oregonstate.edu/courses/1928696/pages/pseudocode-and-flowchart-guidelines) (CLO 4)
* Do [Assignment 3](https://canvas.oregonstate.edu/courses/1928696/assignments/9073286), which gives you practice with controlling flow of execution (CLO 1a-b, MLOs 1-3).
* Write your [Reflection for Project 3a](https://canvas.oregonstate.edu/courses/1928696/assignments/9073298). (CLOs 1a-b, 4, MLOs 1-3).
* Take [Quiz 3](https://canvas.oregonstate.edu/courses/1928696/quizzes/2825532) (CLO 1a-b, MLOs 1-3).

# **Exploration: Comparison operators and logical operators**

## Comparison operators

We often need to compare two values. Python has the following comparison operators:

num\_1 == num\_2 # num\_1 equals num\_2  
num\_1 != num\_2 # num\_1 does not equal num\_2  
num\_1 > num\_2 # num\_1 is greater than num\_2  
num\_1 < num\_2 # num\_1 is less than num\_2  
num\_1 >= num\_2 # num\_1 is greater than or equal to num\_2  
num\_1 <= num\_2 # num\_1 is less than or equal to num\_2

Each of the above **conditions** is either True or False, so if you print out the type of the comparison, you'll see that it's a bool. For example:

17 < 5 # False  
3 != 12 # True

Don't mix up the assignment operator (=) and the equality operator (==). It's a common mistake that can be hard to see.

**You shouldn't test two floats for exact equality because of possible lack of precision or round-off errors.**  Instead, if you need to compare float values for equality, you can do it like this:

abs(num\_1 - num\_2) < 0.000001

This takes the absolute value of the difference between the two numbers and checks whether it's less than a very small value.  **In this course, whenever an assignment requires you to compute a float value, your result will not be tested for exact equality to the expected result - just for equality to within 7 decimal places.**

Unlike many computer languages, Python allows you to **chain** comparison operators.  In Python you check if x is between 2 and 7 by saying

2 < x < 7

whereas in most languages you would need to write that as

2 < x and x < 7

The logical operator **and** is discussed in the following section.

## Logical operators

Sometimes we need to combine comparisons. Python has the following logical operators:

bool\_1 and bool\_2 # Both bool\_1 and bool\_2 equal True  
bool\_1 or bool\_2 # At least one of bool\_1 and bool\_2 equals True  
not bool\_1 # bool\_1 equals False # Flips the value of bool\_1

Each of these conditions is also either True or False.

17 >= 10 and 5 % 2 == 0 # True and False, which equals False  
17 >= 10 or 5 % 2 == 0 # True or False, which equals True  
not 17 >= 10 # not True, which equals False

Assume that P and Q represent two conditions. **P and Q** is only true if P is true **and** Q is true. **P or Q** is true if **either** P is true **or** Q is true (or both). The **not** operator just flips the value from true to false or vice-versa.  So if P is false, then **not P** is true, but if P is true, then **not P** is false.

To check if num\_x and num\_y are both equal to 5, it is **not** correct to do the following:

num\_x and num\_y == 5 # wrong

since that would check whether x is True and y==5 is True.  Instead, you would need to do this:

num\_x == 5 and num\_y == 5

or since we know we can chain comparisons:

num\_x == num\_y == 5

## Exercises

(See the module overview for a link to example solutions.)

1. Print the value of 17 < 5.

Sample input: NA  
Expected Output: False

2. Write code that reads a number from the user, assigns it to a variable, and then prints True if that value is greater than 10, but prints False otherwise.

Sample input: 175  
Expected Output: True

3. Write code that reads a number from the user, assigns it to a variable, and then prints True if that value is between 50 and 100 (inclusive), but prints False otherwise.

Sample input: 50  
Expected Output: True

# **Exploration: If statements**

## if statements

The if statement enables us to define branches in our code based on whether a given condition is true or false. Here's a simple example:

The else case is optional. Both the if and else cases can each contain as many statements as needed. For example:

if direction == 'left':  
 orientation -= 180  
 x\_coord -= 1  
else:  
 orientation += 90  
 x\_coord += 1

You can use and and or to combine tests:

The indentation matters for the program to run correctly. In any computer language indentation is very important for readability, but in most it doesn't affect how the computer executes a program. In Python it's what determines how statements are grouped together. The Python language doesn't force you to indent a specific number of spaces, but it's recommended that you indent 4 spaces unless you're adding to existing code that used something different.

### **nested if statements and elif**

You can also have nested if statements (if statements inside other if statements):

The indentation tells the Python interpreter which ifs and elses go together, however, this level of nesting can get a little hard to read, so the more usual way to do this would be:

Where "elif" is short for "else if".

## Exercises

(See the module overview for a link to example solutions.)

1. Write code that reads a number from the user, assigns it to a variable, and then prints "in range" if that value is between 50 and 100, but prints "out of range" otherwise.

Sample input: 5  
Expected output: "out of range"

2. Write code that reads a number from the user, assigns it to a variable, and then prints "small" if it's less than 10, prints "medium" if it's at least 10 but less than 50, and prints "large" if it's at least 50.

Sample input: 76  
Expected output: "large"

3. Write code that reads an integer from the user and assigns it to a variable. If that integer is a value from 1-5, your program should print out the English word for that number. Otherwise it should print "input not recognized".

Sample input: 4  
Expected output: "four"

# **Exploration: While loops and for loops**

## Iteration

While loops and for loops enable us to make certain sections of code repeat multiple times.

### **while loops**

A while loop continues iterating as long as its condition remains true. Here's an example:

The loop will first test the condition. If it's true, then the code in the loop is executed. It will continue testing the condition and executing the loop until the condition is false, at which point we drop out of the loop and execution resumes with whatever code comes after it. It's important that something in the loop will eventually cause the loop condition to become false. Otherwise the loop will continue forever, which is known as an infinite loop. If you run the example below, you can hit ctrl-c to stop it.

As with if statements, a loop can contain as many statements as needed, with indentation determining how statements are grouped together.

### **for loops**

A for loop traverses a sequence (or **iterable** type), looping once for each element in the sequence. A loop variable takes on the value of the current element, which can then be used inside the loop. So far, strings are the only iterable type we've looked at - we'll see one more on this page and others later. Here's an example of iterating through a string:

The first time through the loop, letter equals "a", next time through it equals "p", next "o", and so on until the last time through, when it equals "y".

### **Using ranges with for loops**

If we need a for loop to iterate over some increasing (or decreasing) sequence of integers, we can achieve that by using a range. Ranges are another iterable type. Here's an example that counts from 1 to 10:

> for num in range(1, 11):  
> print(num)

Notice that the range goes from 1 up to, **but not including**, 11. A range can also count by a given step size:

# counts by twos  
> for num in range(1, 11, 2): # the third number is the step size  
> print(num)

You can count down by using a negative step size, but if you do, then the first number (the start of the range) needs to be greater than the second number (the end of the range):

# counts down from 10 to 1  
> for num in range(10, 0, -1):  
> print(num)

### **break and continue**

A loop can be terminated early with the "break" keyword. For example:

This loop only prints the numbers up to 4 because when the "break" is executed, it immediately drops us out of the loop.

The "continue" keyword allows us to skip iterations of a loop. For example, try replacing the word "break" with "continue" in line 3 of the example above. With that change, it prints the numbers from 1 through 10 except for 5, because it skips that iteration, jumping us to the next iteration of the loop.

### **proper use of break**

There may be occasions where using "break" is justified, but you should try to write your loop conditions such that "break" is not needed, since that usually makes the behavior of your loop simpler to read.

**# bad example**# This code translates a positive decimal integer into binary  
binary\_str = ""  
user\_num = int(input("Please enter a positive integer: "))  
while True:  
 binary\_str = str(user\_num % 2) + binary\_str  
 user\_num //= 2  
 if user\_num == 0: **# This is the real loop condition**  
 break  
print(binary\_str)  
  
**# good example**  
# This code translates a positive decimal integer into binary  
binary\_str = ""  
user\_num = int(input("Please enter a positive integer: "))  
while user\_num != 0:  
 binary\_str = str(user\_num % 2) + binary\_str  
 user\_num //= 2  
print(binary\_str)

Notice that we flipped from testing == to testing !=. That's because we were testing when to stop, but now we're testing when to keep going.

**# bad example**  
# This code continues asking the user for integers and printing them out  
# until the user enters one that is a multiple of 10  
while True:  
 user\_num = int(input("Please enter an integer: "))  
 if user\_num % 10 == 0: **# This is the real loop condition**  
 break  
 else:  
 print("You entered", user\_num)  
print("all done")  
  
**# good example**  
# This code continues asking the user for integers and printing  
# them out until the user enters one that is a multiple of 10  
  
# The first line declares user\_num by initializing it to a value  
# that will make the loop condition True (otherwise we would  
# never enter the loop)  
user\_num = 1  
while user\_num % 10 != 0:  
 user\_num = int(input("Please enter an integer: "))  
 print("You entered", user\_num)  
print("all done")

If there are two things that need to be true for a loop to continue, it's easy enough to use **and** to connect them in the loop condition.   You can also do this when there are more than two things that need to be true, but it's possible in such cases that the loop condition could become as hard or harder to read than using break statements.

### **nested loops**

Both while loops and for loops can be nested. You can also have if statements inside loops or loops inside if statements. You can have an if statement inside a for loop inside a while loop inside another if statement if you want to.

## Exercises

(See the module overview for a link to example solutions.)

1. Write code that reads an integer from the user and prints out the sum of the integers from 1 to that number.

Sample input: 3  
Expected output: 6

2. Write code that reads a string from the user, counts how many characters are in the string, and prints out "odd" if that number is odd, but prints "even" if that number is even.

Sample input: The harder you're thrown, why the higher you bounce  
Expected output: "odd"

3. Write code that continues reading a string from the user and printing it out until the user enters "quit".

Sample input:   
 Pessimism = sin against science.  
 Energy out of the tin, climate control, perpetual growth!  
 To convert Iceland into a tropical paradise –  
 no problem. The rest makes no difference.  
 quit  
Expected output:  
 Pessimism = sin against science.  
 Energy out of the tin, climate control, perpetual growth!  
 To convert Iceland into a tropical paradise –  
 no problem. The rest makes no difference.

# **Exploration: Code tracing**

## Flow of Execution: Code Tracing

Being able to trace the flow of execution through a given piece of code and tell how it will behave, is a critical skill for reading and understanding code, as well as for debugging. Here are four common methods people use to help them do this:

1. Hand-tracing is just stepping through the code as if you were the computer, going along line by line, and keeping track of the current values of all the variables as they change. You can write down on paper (or a whiteboard) a row (or column) for each variable, and then each time the value for that variable changes, you can cross off the old value and write down the new value.
2. Temporarily inserting print statements at various places in your code can also be useful for checking what values variables referred to at those points. Make sure to delete such print statements when you're done with them.
3. Many programming environments provide a built-in debugger, which lets you look at the current values of the different variables to see how they change as you step through the code line by line. PyCharm includes such a debugger. Here's a [tutorial Links to an external site.](https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html)on how to use it. It's not required for this course, but you'll probably find it very helpful.
4. Explaining how your code works to someone else can often give you an "Aha!" moment where it helps you see a disconnect between what your code is supposed to do and what it actually does. This can even work when explaining your code to a cat or dog, or even a rubber duck.

All of these are worth being comfortable with.

## Exercise

Go through the [PyCharm debugger tutorial Links to an external site.](https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html). No, really, I mean it. You'll thank me later.

# **Module 3 - Animation and Video Demos**

## flow of execution

This video demonstrates the use of branching (if statements) and iteration (loops) to control the flow of execution in a program that tests whether a number entered by the user is prime.

[View video in separate window](https://media.oregonstate.edu/media/t/0_7xto32hy)

## comparison operators and logical operators

## if statements

## while loops and for loops

[Previous](https://canvas.oregonstate.edu/courses/1928696/modules/items/22739811)[Next](https://canvas.oregonstate.edu/courses/1928696/modules/items/22739813)

# **Additional examples for Module 3 - part 1**

## Tinker Lab

### **Code Sample 1**

What does the following code do?

What do the **>** and **<** operators do?

What do the **and** and **or** operators do?

What does the **==** operator do? How is it different from the **=** operator?

What does the **if** keyword do?

Indentation is important in Python! What is it doing here?

### **Code Sample 2**

This version does the same thing:

What are the differences from the first version?

What does the **!=** operator do?

What effect does nesting three of the if statements inside the other one have?

What does the **else** keyword do?

### **Code Sample 3**

Here's a third version:

What is the difference from the previous version?

The **elif** keyword is short for "else if" - what does it do?

Why do we no longer need to have an explicit check for whether any of the numbers are equal?

Modify this code (or one of the other versions) so that if all number choices are different it behaves the same, but if exactly two players choose the same number, then the other player wins. If all three players choose the same number then no one wins.

# **Additional examples for Module 3 - part 2**

## Tinker Lab

### **Code Sample 1**

What does the following code do?

What does the **while** loop do?

What does the indentation signify?

What does the **+=** operator do?

Modify this code to keep track of the product of the numbers instead of their sum.

### **Code Sample 2**

What does the following code do?

What does the **for** loop do?

What happens if you change "ch + rev\_name" to "rev\_name + ch"? Why?

### **Code Sample 3**

What does the following code do?

What does a for loop with a **range** do?

Why do you think we're adding 1 to upper\_limit when defining the range? (We don't have to do that, but the results would be different.)

### **Code Sample 4**

How does the behavior of this version differ?

What is the effect of having the if statement indented inside the loop?

What is the if statement doing?

### **Code Sample 5**

Here's another version that does the same thing:

Modify both versions to add the even values instead.

Modify both versions to add every third value up to the upper\_limit, starting with 2.

# **Help Session for Week 3**

## Note

Help Sessions or Webinars in this course are neither mandatory nor graded based on attendance. They are completely optional.

More info at  [https://edstem.org/us/courses/31713/discussion/2376398Links to an external site.](https://edstem.org/us/courses/31713/discussion/2376398)

## Video recording:

**Note:** Don't trust the captions too much. The video is auto-transcribed and Zoom doesn't really do a great job of understanding my accent. Please feel free to ask me for clarifications if you have any questions :)

## Tools we used

The only tool we used during this Help Session for demo:

* [https://pythontutor.com/python-debugger.html#mode=edit Links to an external site.](https://pythontutor.com/python-debugger.html#mode=edit)
  + To learn how to use this, try and use the different buttons on the site :) You won't break the tool.

## Recommended watching

A video I reference during this Help Session is[https://youtu.be/Ct-lOOUqmyYLinks to an external site.](https://youtu.be/Ct-lOOUqmyY)[Shape, arrow

Description automatically generated](https://youtu.be/Ct-lOOUqmyY)

This video shows how a computer is (most probably) going to interpret your instructions and what problems imprecise instructions can lead to.

## Code samples

* [Example of a for loopLinks to an external site.](https://pythontutor.com/visualize.html#code=repetition%20%3D%20input%28%22Enter%20number%20of%20repetitions%22%29%0Arepetition%20%3D%20int%28repetition%29%20%23convert%20the%20input%20string%20to%20an%20integer%0A%0A%23repeatedly%20execute%20as%20many%20time%20as%20we%20have%20numbers%20%0A%23in%20the%20range%20from%200%20upto%20repetition%20%0A%23and%20during%20each%20repetition%20put%20that%20number%20inside%20the%20variable%20i%0A%0Afor%20i%20in%20range%280,%20repetition%29%3A%20%0A%20%20%20%20print%28%22This%20is%20iteration%20%22,%20i%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%222%22%5D&textReferences=false)
* [Example of an if-else statementLinks to an external site.](https://pythontutor.com/visualize.html#code=%23if%20I%20have%20milk,%20then%20I%20will%20make%20coffee.%20%0A%23If%20I%20don't%20have%20milk,%20then%20I%20will%20make%20green%20tea.%0Amilk%20%3D%20True%0A%0Aif%20milk%20is%20False%3A%0A%20%20%20%20%23make%20coffee%0A%20%20%20%20print%28%22Let's%20make%20coffee%22%29%0Aelse%3A%0A%20%20%20%20%23make%20tea%0A%20%20%20%20print%28%22Let's%20make%20green%20tea%22%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)
* [Visualizing Code Sample 2 from Additional Examples - Part 2Links to an external site.](https://pythontutor.com/visualize.html#code=name%20%3D%20input%28'Please%20enter%20your%20full%20name%3A%20'%29%0Arev_name%20%3D%20''%0Afor%20ch%20in%20name%3A%20%20%0A%20%20%20%20rev_name%20%3D%20ch%20%2B%20rev_name%0A%0Aprint%28rev_name%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%22ABC%22%5D&textReferences=false)
* [What happens when you use break with the above code sampleLinks to an external site.](https://pythontutor.com/visualize.html#code=name%20%3D%20input%28'Please%20enter%20your%20full%20name%3A%20'%29%0Arev_name%20%3D%20''%0Afor%20ch%20in%20name%3A%20%20%0A%20%20%20%20rev_name%20%3D%20ch%20%2B%20rev_name%0A%20%20%20%20break%0Aprint%28rev_name%29%0A&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%22ABC%22%5D&textReferences=false)
* [What happens when you change the order of operands/variables around the + operator in Code Sample 2 from Additional Examples - Part 2Links to an external site.](https://pythontutor.com/visualize.html#code=name%20%3D%20input%28'Please%20enter%20your%20full%20name%3A%20'%29%0Arev_name%20%3D%20''%0Afor%20ch%20in%20name%3A%20%20%0A%20%20%20%20rev_name%20%3D%20rev_name%20%2B%20ch%0Aprint%28rev_name%29&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%22ABC%22%5D&textReferences=false)

## How to use these code samples (or pythontutor.com in general)?

* Click on "Visualize execution" and then use the "next" and "prev" button to run the program step by step.
* As you run through the program step by step, you will see variables from your program on the right hand side of the screen changing. Note how each operation affects the variable.
* You can enter input, when prompted to do so, below the code editor window.
* The output appears above the variables.
* Try changing the code using "Edit this code" link. Try different variations by asking questions like "What if I put a different initial value for this variable?", "What if I switch the variables around this operator?", "What if I change the order of lines in this code?"
* I promise you that you won't hurt your computer by doing anything with this tool ;) The code is executed on their server and not your machine.
* If at any point, the code is "broken" or you want to go back to the original version come back to this Canvas page and click on the code link.
* You can use PythonTutor.com for any code samples that you come across in this course or even your own assignment code to debug it.

## Questions?

Do one of these:

1. Ask in the Ed Discussions thread for this Help Session.
2. Ask in General channel on Teams
3. Schedule an office hour with me using the info at https://tinyurl.com/hedaoos

## Bonus

I did a webinar in Winter 2021 demonstrating loops and conditionals statements in Python on a robot.

You can watch the recording at <https://media.oregonstate.edu/media/t/1_ug4aattc> and find the code samples and slides at [this Google Slides link Links to an external site.](https://docs.google.com/presentation/d/1TvITJdeb8KXZPV9GJn95YSicsIW9DEt7cSYF05uGEFM/edit#slide=id.g86e7dc082d_0_5)(Only accessible when you are logged in using your OSU account)

# **Pseudocode and flowchart guidelines**

## 

Adapted from John Dalbey, Bob Roggio and Farshad Barahimi (CS 165).

Pseudocode and flowcharts allow the designer to focus on the logic of the algorithm without being distracted by details of language syntax. At the same time, the pseudocode or flowchart needs to be complete. It describes the entire logic of the algorithm so that implementation becomes a matter of translating from the pseudocode or flowchart into source code.

## Pseudocode

In general the vocabulary used in the pseudocode should be the vocabulary of the problem domain, not of the implementation domain. The pseudocode is a narrative for someone who knows the requirements (problem domain) and is trying to learn how the solution can be organized. That is, someone who is a non-programmer should be able to read and understand your pseudocode. For example:

Extract the next word from the line (good)  
Set word to get next token (poor)

Append the file extension to the name (good)  
Name = name + extension (poor)

For every character in the name (good)  
For character = first to last (poor)

Note that the logic for each line must be at the level of a single loop or decision. Thus "Search the list and find the customer with highest balance" is too vague because it takes a loop **and** a nested decision to implement it.

Each textbook and each individual designer may have their own personal style of pseudocode. Pseudocode is not a rigorous notation, since it is read by other people, not by the computer, but **this is the style you should follow for this class.**

### **Examples:**

Get student's grade from user

If student's grade >= 60

Print "passed"

else

Print "failed"

Set total to 0

Set grade counter to 1

While grade counter <= 10

Input the next grade

Add the grade to the total

Add 1 to the grade counter

Set the class average to the total divided by 10

Print the class average

## Flowcharts

A flowchart depicts an algorithm in a graphical format. Here are descriptions of the most common symbols:

* Oval: start and end
* Arrow: represents flow of control in a program.
* Rectangles: shows a computation or a specific process, for example "multiply X by 2".
* Parallelogram: Used for getting **input** from user or sending **output** to user.
* **Diamond**: Used for conditional flow control where a program has to decide which way to go, for example: if X is divisible by 2 do this thing; if not, do this other thing. Without conditional flow control, a program would have just one path from start to end, but with it there may be many different paths from start to end.

### **Examples:**