

PYTHON KEYWORDS

while Exercises

Python Keywords

REFERENCE · WHILE · EXERCISES

💻 Practice: Looping While a Condition Holds

The companion exercise set for the `while` reference page. Nine exercises (the ten-exercise limit is a maximum, and these cover what the keyword genuinely needs): taking numbers apart, a sequence that ends when it ends, a goal that is reached after an unknown number of months, a flag-controlled validation loop, a stack, binary search, Euclid's method, the walrus operator, and a compound condition with no `break`. Try each one yourself before opening its solution. Every solution was run, and its sample output is the real output.

Exercise 1: Reverse the Digits of a Number

Given `n = 12345`, build the number with its digits reversed (`54321`) using only a `while` loop and arithmetic: no `str()`, no slicing. Then try `1200` and explain what happens to the zeros.

Goal: Use a `while` loop whose number of passes depends on the data, taking digits off with `% 10` and `// 10` and building the result with `* 10`.

→ Solution

Exercise 2: The Collatz Sequence

Start from `n = 6`. While `n` is not 1: if it is even, halve it; if it is odd, multiply by 3 and add 1. Record every value, then print the whole sequence joined by `->` and the number of steps taken. Then try `n = 27` and print only the step count.

Goal: The classic case of a `while` loop where nobody knows in advance how many passes it needs: the condition is the only thing that ends it.

→ Solution

Exercise 3: When Will the Savings Reach the Goal?

You start with 0, pay in 200 at the end of every month, and the balance earns 0.5% interest per month, added before the deposit. How many months until the balance reaches at least 5000? Print the number of months, as years and months, and the final balance.

Goal: Use the loop condition itself (`balance < goal`) as the stopping rule, with no `break`, and make sure the balance is updated inside the loop.

→ Solution

Exercise 4: A Password Chooser That Names the Rules You Broke

Keep asking for a password until it follows all of these rules: at least 8 characters, at least one digit, and no spaces. After each bad attempt, list exactly which rules were broken. Use a variable such as `accepted` in the `while` condition, not `while True` and `break`.

Goal: Control a `while` loop with a flag, build a list of problems on each pass, and report them all at once.

→ Solution

Exercise 5: Flatten a Nested List Without Recursion

Given `nested = [1, [2, [3, 4]], 5, [[6], 7]]`, produce the flat list `[1, 2, 3, 4, 5, 6, 7]` in the original order, using a `while` loop and a list used as a stack. No recursion and no function that calls itself.

Goal: Use `while stack:` (a non-empty list is true) to work through items until nothing is left, where processing an item can put more items on the stack.

→ Solution

Exercise 6: Binary Search in a Sorted List

Given `numbers = [3, 8, 15, 23, 42, 57, 61, 90]`, search for each of the targets `42, 50, 3, 90`, printing either `found at index N` or `not found`. Do not use `in`, `.index()` or a `for` loop over the list for the search: halve the search range on every pass.

Goal: A `while` loop whose condition (`low <= high`) means "there is still somewhere to look", with the range narrowed on every pass.

→ Solution

Exercise 7: Euclid's Greatest Common Divisor

For each pair in `[(48, 18), (1071, 462), (17, 5), (100, 25)]`, print the greatest common divisor using Euclid's method: while the second number is not zero, replace the pair `(x, y)` with `(y, x % y)`. The answer is the first number when the loop ends.

Goal: Update two variables at once with tuple assignment, and let the condition `y != 0` end the loop when the remainder reaches zero.

→ Solution

Exercise 8: Average of Numbers Until a Blank Line

Ask for numbers until the user presses Enter on an empty line. Then print how many numbers were entered and their average to two decimal places, or `No numbers entered` if there were none. Read the input with the walrus operator inside the `while` condition, so there is no second `input()` call and no `break`.

Goal: Use `:=` to assign and test in one step: `while (text := input(...)) != "":`.

→ Solution

Exercise 9: A PIN Check With Three Attempts and No break

Ask for a PIN (the correct one is `1234`) with at most three attempts. Show how many attempts are left in the prompt. Print `Welcome` after a correct PIN, or `Card retained` after three

wrong ones. Use a single `while` condition that combines the attempt limit and a success flag, and do not use `break`.

Goal: Write a compound condition with `and`, and make sure both the counter and the flag are updated inside the loop so that either one can end it.

→ Solution