Python While Loops

Table of contents

Augmented assignment operators 1
`break` 2
Simulating a do-while 2
User input 4
Random number generation 4

A **loop** is a control structure that repeats code that belongs to it. A **while-loop** is a loop that repeats code *while some condition is satisfied*:

```
while <condition>:
    <code>
```

The condition is checked before every repetition (including the first) — if it's false to begin with, the loop body never runs at all:

```
>>> x = 0
>>> while False:
...     print(x)
...     x += 1
>>> # nothing prints
```

See the while-loop control-flow figure in 2025-08-18-while-loops[1] for a picture of this.

Augmented assignment operators

[1] courses/csse1001/2025-08-18-while-loops.pdf

Operator	Equivalent to
`x += y`	`x = x + y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`

These are the four used in the course. They are what makes an accumulator loop readable:

```
>>> x = 0
>>> while x < 10:
...     x += 1
>>> x
10
```

`break`

The `break` keyword terminates the (innermost) loop immediately and continues with the rest of the program — so code after it *inside the loop body* is unreachable:

```
>>> while True:
...     print("hello")
...     break
...     print("world")
hello
```

Simulating a do-while

A **do-while** (or **repeat-until**) is a while-loop variant available in other languages but not natively supported in Python. It runs its code *at least once*, because the condition is only tested after the body:

```
do:
    <code>
while <condition>   # not valid Python syntax
```

Although not natively supported in Python, we can simulate one with `while True` and a `break` (some programmers prefer `1` to `True`):

[2] courses/csse1001/2025-08-18-while-loops.pdf

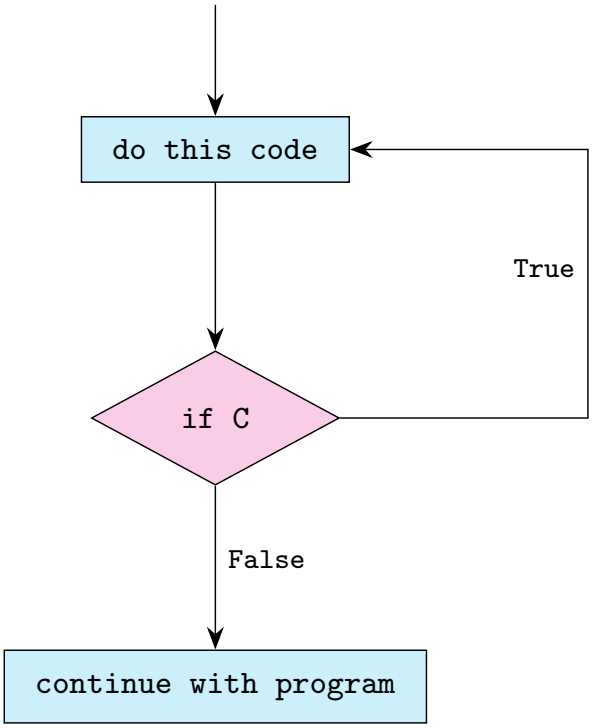

Figure 1: The do-while as a flowchart. Compare the while-loop figure in 2025-08-18-while-loops[2]: the body is above the test rather than below it, so the body always runs once before `C` is ever evaluated, and the `True` edge is the one that loops back.

```
>>> x = 0
>>> while 1:
...     x += 1
...     if x > 0:
...         break
>>> x            # x retains its value outside the while
1
```

`x` ends at `1`, not `0`: the body ran once before the exit test was reached, which is the whole point of the shape.

User input

`x = input()` waits for input from the keyboard and assigns it to `x`, always with `type(x) is str`. Numeric input therefore has to be converted — `int(input("Guess: "))`.

Random number generation

Random number generation is handled by an external library. It is **not** built-in and therefore must be imported. Importing the single command:

```
>>> from random import randint
>>> randint(1, 6)   # chosen uniformly over the interval
2
```

or the entire library:

```
>>> import random
>>> random.randint(1, 6)
2
```

Both endpoints are included: `randint(1, 6)` is the course's six-sided die, and its prompt reads `Guess in [1..6]`.