# Python While Loops

## Table of contents



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
```

### Augmented assignment operators

| Operator | Equivalent to |
|---|---|
| `x += y` | `x = x + y` |
| `x *= y` | `x = x * y` |
| `x /= y` | `x = x / y` |

| Operator | Equivalent to |
|---|---|
| `x %= y` | `x = x % y` |

**`break`**

The `break` keyword terminates the (innermost) loop immediately and continues with the rest of the program:

```
>>> 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, that runs its code *at least once* before checking the condition:

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

We can simulate one with `while True` (some programmers prefer `1` to `True`) and a `break`:

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

## User input

`x = input()` waits for input from the keyboard and assigns it to `x`, always with `type(x) is str`.

## Random number generation

Random number generation is handled by an external library (not built-in), so it must be imported:

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

or, importing the entire library:

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