The while Loop

This lesson highlights the key features of the 'while' loop.

We'll cover the following

The `while` loop keeps iterating over a certain set of operations as long as a certain **condition** holds `True`.

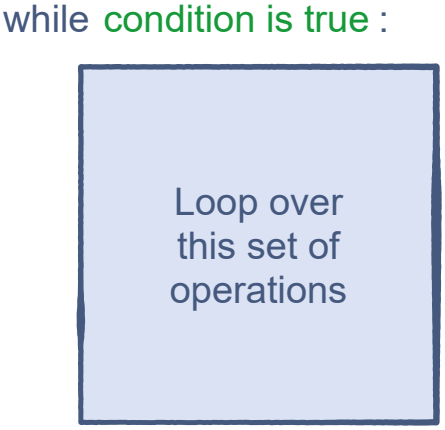
It operates using the following logic:

While this condition is true, keep the loop running.

Structure

In a `for` loop, the number of iterations is fixed since we know the size of the sequence.

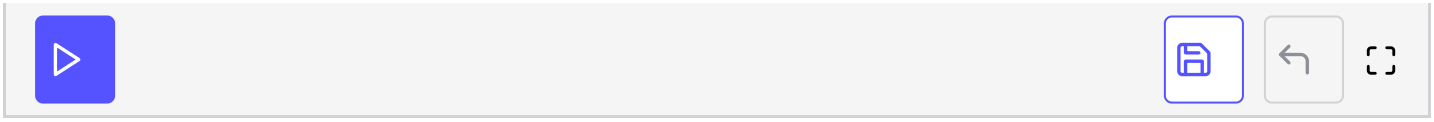
On the other hand, a `while` loop is not always restricted to a fixed range. Its execution is based solely on the condition associated with it.



The `while` Loop in Action

Here's a `while` loop that finds out the maximum power of `n` before the value exceeds `1000`:

```
n = 2  # Could be any number
power = 0
val = n
while val < 1000:
    power += 1
    val *= n
print(power)
```



In each iteration, we update `val` and check if its value is less than `1000`. The value of `power` tells us the maximum power `n` can have before it becomes greater than `1000`. Think of it as a counter.

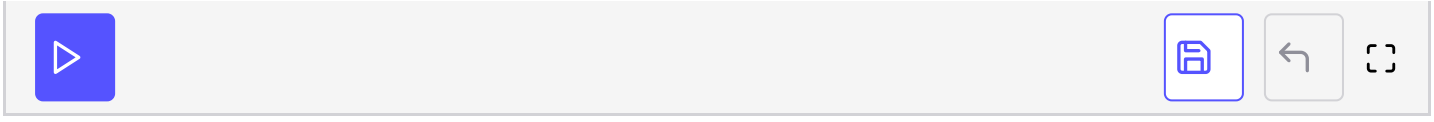
We can also use `while` loops with data structures, especially in cases where the length of data structure changes during iterations.

The following loop computes the sum of the first and the last digits of any integer:

```
n = 249
last = n % 10  # Finding the last number is easy

first = n  # Set it to `n` initially
while first >= 10:
    first //= 10  # Keep dividing by 10 until the leftmost digit is reached.

result = first + last
print(result)
```



Cautionary Measures

Compared to `for` loops, we should be more careful when creating `while` loops. This is because a `while` loop has the potential to never end. This could crash a program!

Have a look at these simple loops:

```
while(True):
```

```
    print("Hello World")

x = 1

while(x > 0):
    x += 5
```

The loops above will never end because their conditions always remain true. Hence, we should always make sure that our condition has a mutable variable/object that is being updated in the loop and will eventually turn the condition false.

Other Properties

The `break`, `continue`, and `pass` keywords work with `while` loops.

Like `for` loops, we can also nest `while` loops. Furthermore, we can nest the two types of loops with each other.

Iteration vs. Recursion

If we observe closely, there are several similarities between iteration and recursion. In recursion, a function performs the same set of operations repeatedly but with different arguments.

A loop does the same thing except that the value of the iterator and other variables in the loop's body change in each iteration.

Figuring out which approach to use is an intuitive process. Many problems can be solved through both.

Recursion is useful when we need to divide data into different chunks. Iteration is useful for traversing data and also when we don't want the program's scope to change.

In the next section, we'll explore the different **data structures** of Python. Before that, be sure to check out the quiz and exercises at the end of this section to test what you've learned here.