Lambdas

In this lesson, we'll study a unique category of functions called lambda.

We'll cover the following

So far, we've always given names to our functions using the `def` keyword. However, there is a special class of functions for which we do not need to specify function names.

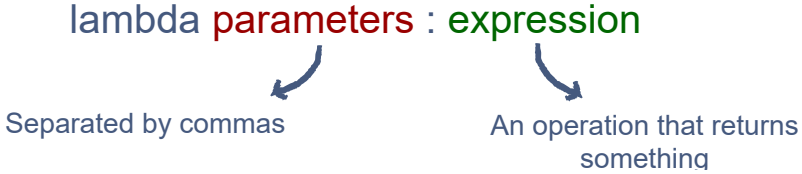
Definition

> A **lambda** is an anonymous function that *returns* some form of data.

Lambdas are defined using the `lambda` keyword. Since they return data, it is a good practice to assign them to a variable.

Syntax

The following syntax is used for creating lambdas:

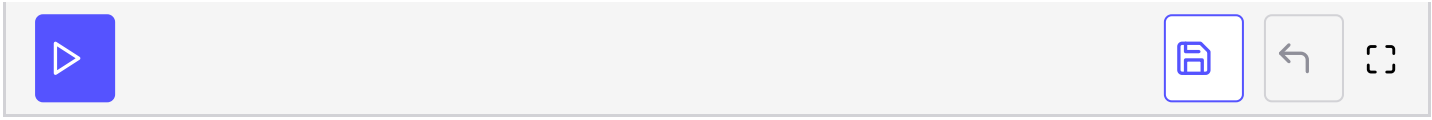


In the structure above, the `parameters` are optional.

Let's try creating a few simple lambdas.

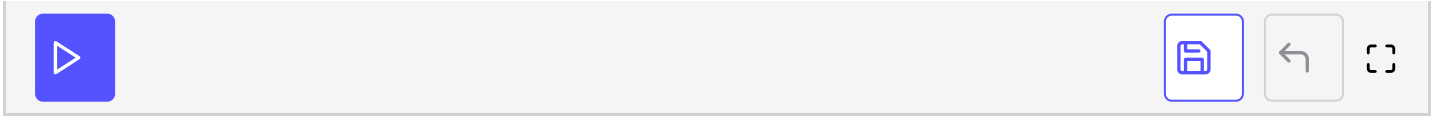
Below, we can find a lambda that triples the value of the parameter and returns this new value:

```
triple = lambda num : num * 3  # Assigning the lambda to a variable

print(triple(10))  # Calling the lambda and giving it a parameter
```



Here's a simple lambda that concatenates the first characters of three strings together:

```
concat_strings = lambda a, b, c: a[0] + b[0] + c[0]

print(concat_strings("World", "Wide", "Web"))
```



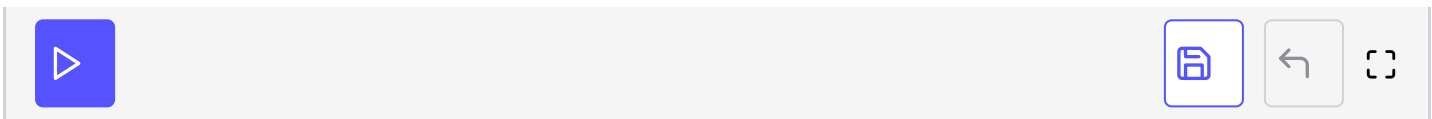
As we can see, lambdas are simpler and more readable than normal functions. But this simplicity comes with a limitation.

A lambda cannot have a multi-line expression. This means that our expression needs to be something that can be written in a single line.

Hence, lambdas are perfect for short, single-line functions.

We can also use conditional statements within lambdas:

```
my_func = lambda num: "High" if num > 50 else "Low"

print(my_func(60))
```



The Purpose of Lambdas

So, what is the point of having lambdas around? We're still assigning them to variables, so they do have names.

They can be written in-line, but that isn't a huge advantage.

Well, lambdas are really useful when a function requires another function as its argument.

This concept will be explored in the next lesson where we'll understand the true purpose of lambda functions.