Lambdas

In this lesson, we will learn about lambdas and why they are important.

WE'LL COVER THE FOLLOWING

There is a special class of functions for which we do not need to specify function names. These are called lambdas.

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 good practice to assign them to a variable.

Syntax

```
lambda parameters: expression
```

In the above syntax, `parameters` are optional.

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

```
square = lambda num: num ** 2 # Assigning the lambda to a variable
print (square(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]
```

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

print (concat_strings("Quantum", "Information", "Science"))
```

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.

> 💡 lambdas are really useful when a function requires another function as its argument.

Calling functions with lambdas as arguments

In Python, one function can become an argument for another function.

Using lambda functions, let's make a `calculator` function that has custom functionality along with two numbers as arguments.

```
def calculator (operation, n1, n2):
  return operation(n1, n2) # Using the 'operation' argument as a function

# 10 and 20 are the arguments.
# The lambda multiplies them.
result = calculator(lambda n1, n2: n1 * n2, 10, 20)

print (result)

# 10 and 20 are the arguments.
# The lambda adds them.
print (calculator(lambda n1, n2: n1 + n2, 10, 20))

# 10 and 3 are the arguments.
# The lambda computes 10^3
print (calculator(lambda n1, n2: n1 ** n2, 10, 3))
```

The code looks much cleaner with a lambda function. We can define the operation on the go whenever we want.

In the next lesson, we'll learn about some important data structures in Python.