Functions

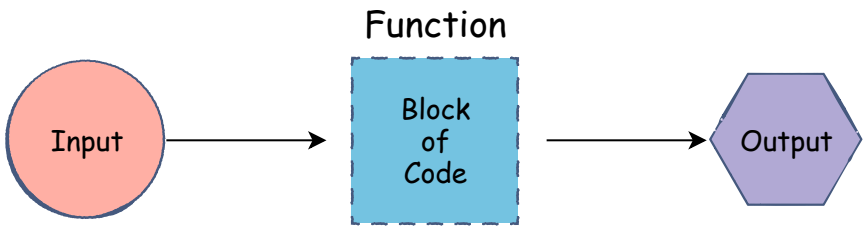
This lesson will introduce functions in Python.

WE'LL COVER THE FOLLOWING

Function

A **function** in a programming language is a reusable block of code that will be executed when called. You can pass data to it and it can return data as well.

Think of a function as a box that performs a task. We give it an input and it returns an output. We don't need to write the code again for different inputs, we could just call the function again.



Until now, we have only been using `print` and `len`. Both of these are functions. They perform a specific predetermined task. These are called **built-in** functions as they come with the Python language. Similarly, there are many other functions that come built-in.

Let's say we want to find the minimum of two numbers.

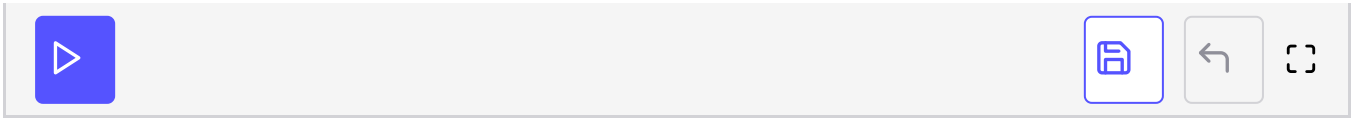
```
num1 = 10
```

```
num2 = 40
# code to check minimum number
if num1 < num2:

  minimum = num1
else:
  minimum = num2
print(minimum)

num1 = 250
num2 = 120
# code to check minimum number
if num1 < num2:
  minimum = num1
else:
  minimum = num2
print(minimum)

num1 = 100
num2 = 100
# code to check minimum number
if num1 < num2:
  minimum = num1
else:
  minimum = num2
print(minimum)
```



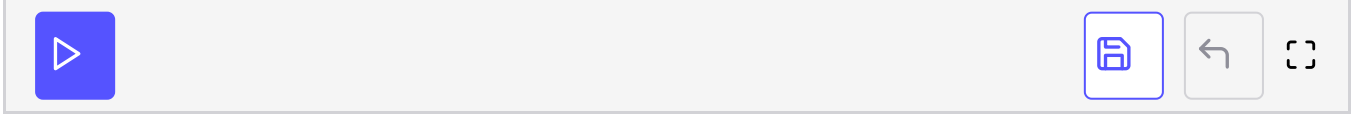
For every new pair of integers, we need to write the `if-else` statement again. This could become much simpler if we had a function to perform the necessary steps for calculating the minimum. The good news is that Python already has the `min` function:

```
# find minimum using function
minimum = min(10,40)
print(minimum)

# find minimum using function
minimum = min(250,120)
print(minimum)

# find minimum using function
minimum = min(100, 100)
print(minimum)

# It even works with multiple arguments
minimum = min(100,120,34,56)
print(minimum)
```



We use the `min` function just like we use `print` and `len`; we pass in a list of

values separated by commas.

This was an example of using **built in** functions. But we can also create and use our own functions.

Creating Functions

In Python, a function can be defined using the `def` keyword in the following format:

```
def function_name (arguments):
    body
```

The `function_name` is simply the name we will use to identify the function.

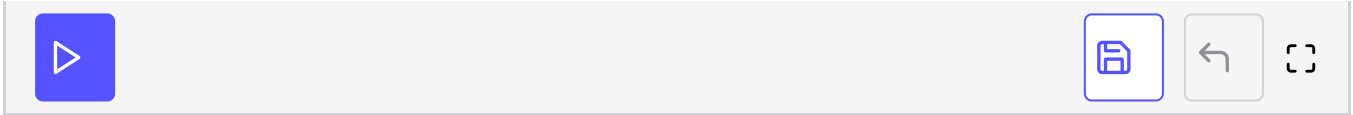
The `arguments` of a function are the inputs for that function. We can use these inputs within the function. Arguments are optional. We'll get to know more about these later.

The `body` of the function contains the code for the task that we want the function to perform. This is always indented to the right.

Let's create our first function.

```
def my_print_function(): # No arguments
  print ("This")
  print ("is")
  print ("a")
  print ("function")
# Function ended

#Calling the function in the program multiple times
my_print_function()
my_print_function()
```



In **line 1**, we use the keyword `def` to define our function. Next we write the name of the function. We name it `my_print_function`, then we use parenthesis. But we leave these empty as we do not need any arguments for this function.

In **lines 9 and 10**, we simply use this function. Using a function is known as **calling** a function. We write the name of the function with parenthesis. In this

way, we do not need to write the same code again and again.

Function Arguments

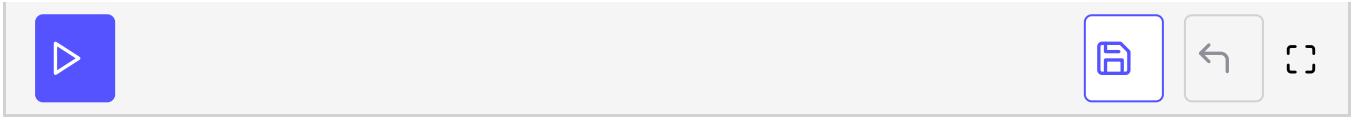
Arguments are a crucial part of the function structure. They are the means of passing data to the function. This data can be used by the function to perform a meaningful task.

When creating a function, we must define the number of arguments and their names. These names are only relevant to the function and will not affect variable names elsewhere in the code. Arguments are enclosed in parentheses and separated by commas.

Continuing our discussion from the last example, where we had to write the same piece of code twice, we will write a function that will print something on the basis of the argument it gets.

```
def color_print (color): # two arguments
    if color=='red':
        print("Red is my color")
    elif color == 'yellow':
        print("Yellow is my color")
    elif color == 'black':
        print("Yellow is my color")
    else:
        print("I dont know the color")

color_1 = 'red'
color_2 = 'black'
# call function with arguments
color_print(color_1)
color_print(color_2)
```



In **line 1**, we use the keyword `def` to define our function. Next, we write the name of the function. We name it `color_print`, then we use parenthesis. We write the argument name. Then in **lines 2-9**, we write the code for printing a message on the basis of the argument we received.

In **lines 11 and 12**, we create two variables, `color_1` and `color_2`, and give them some values. We call our `color_print` function in **lines 14 and 15**. We give `color_1` and `color_2` as arguments to the functions on **lines 14 and 15** respectively.

We see that now we do not have to write the same piece of code again and again. This is why functions are always used in programming.

The `return` Statement

So far, we've only defined functions that print something. They don't give anything back to us. But if we think back, functions return values all the time. Just take `len` for example. It returns an integer that is the length of the list passed to it.

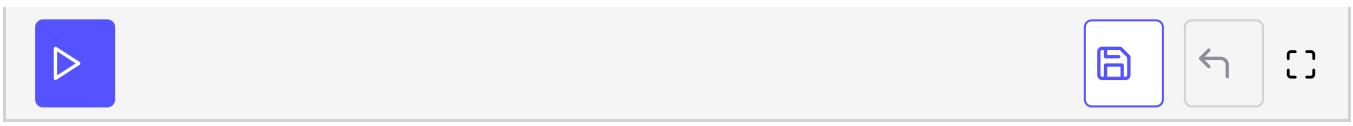
To return something from a function, we must use the `return` keyword. Keep in mind that once the `return` statement is executed, the function ends. Any remaining lines of code after the `return` statement will not be executed.

We saw the `min` function above. Let's write our own version of the function `minimum` to return the smaller value of two numbers. It will work just like the built-in `min` function with two parameters:

```
def minimum (first, second):
    if (first < second):
        # exit function returning value
        return first
    else:
        # exit function returning value
        return second

num1 = 10
num2 = 20

result = minimum (num1, num2) # Storing the value returned by the function
print (result)
```



We have defined the `minimum` function in the same way as we defined the `color_print` function above. In **lines 4 and 7**, we return the number by using the `return` statement.

In **line 12**, we call the function. The positions of the parameters are important. In the case above, the value of `num1` will be assigned to `first` as it was the first parameter. Similarly, the value of `num2` is assigned to `second`. We store the returned value in a variable called `result`. We print `result` in the last

the returned value in a variable called `result`. We print `result` in the last line.

So now you can write your own functions in Python. In the next lesson, we will look at another fundamental concept, *lists*, in Python.