What is Recursion?

In this lesson, we will introduce recursion.

We'll cover the following

This chapter gives a basic introduction to recursion. If you feel confident about your knowledge of recursion, feel free to skip to a further lesson in the chapter. If you want to review or practice recursion, you can attempt the challenges in this chapter.

What is the deal with recursion?

If you're in Tech, chances are you've heard a lot about recursion, so much so that it starts to sound intimidating. Well, we have you covered with this simple definition for recursion:

> *Recursion* is the process of repeating a procedure.

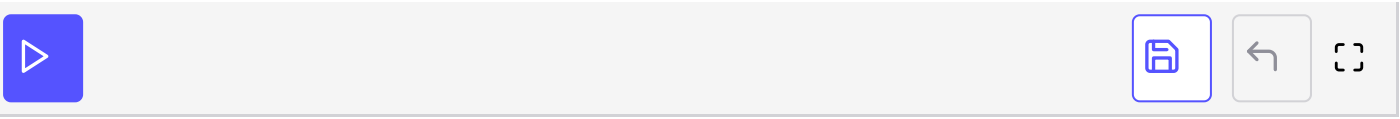
Yes! That's it. In computer science, recursion refers to a function calling itself, directly or indirectly. We will go over a couple of examples in this lesson before moving on to more difficult challenges later.

Direct recursion

As the name suggests, direct recursion happens when a function calls itself directly in its body or definition. Look at the following code as an example:

```
def func(str , n):
  if n > 0:
```

```
    print(str, "called func with n =", n)
    func("func", n-1)


def main():
  func("main" , 7)

main()
```



We can see how `func` makes a recursive call to itself in *line 4*. Also, look at the `if` condition in *line 2*; this is called the base case. If we were to remove this line, our program would not know when to stop execution and Python will throw an error saying:

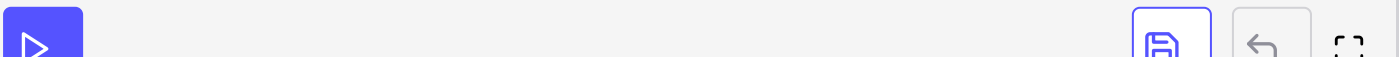
```
RecursionError: maximum recursion depth exceeded while calling a Python object
```

The order of *lines 3 and 4* is important here. If you were to change their order you would see how output order also changes so numbers are printed in descending order. Therefore, while this ordering might seem trivial; it can make all the difference. The first case, where the `print` statement occurs before the recursive call, is called **tail recursion**. This is because the recursive call is following the processing, `print` in this case. Whereas in the second case, when the number is displayed after the recursive call, it is called **head recursion** because the recursive call occurs at the start of the function before any other statement.

Indirect recursion

When two or more functions call themselves indirectly from each other, we call it **indirect recursion**. Look at the following code playground:

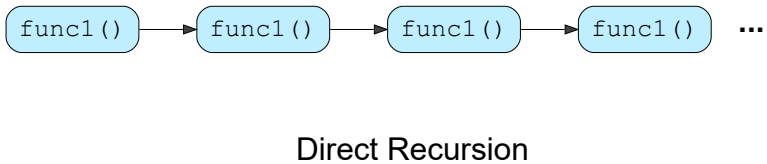
```
def func1(str, n):
  if n > 0:
    print (str, "called func1 with n =", n)
    func2("func1", n-1)

def func2(str, n):
  if n > 0:
    print (str, "called func2 with n =", n)
    func1("func2", n-2)

def main():
  func1("main", 7)
main()
```



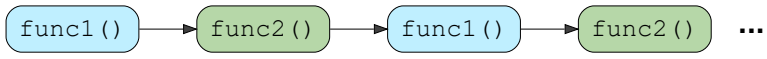
Nothing tricky here: `func1` calls `func2` with `n-1` in *line 4*, whereas `func2` calls back `func1` with `n-2` in *line 9* until `n` becomes less than or equal to `0`. Following is the order of the function calls:

```
func1 -> func2 -> func1 -> func2
```

Look at the figure below for a comparison of direct and indirect recursion.



Direct Recursion



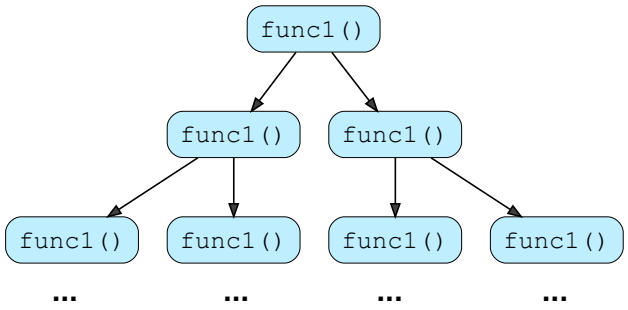
Indirect Recursion

Different types of Linear recursion

Non-linear recursion

Both the examples we discussed had a single call to themselves; This type of recursion is called *linear* recursion. You might have a case where a function calls itself multiple times instead of once; we call that *non-linear* recursion. Most of the recursion we will deal with in this course will be non-linear. One famous example of non-linear recursion is the *Fibonacci series* algorithm. It is a *binary recursion* since at each step the function recurs twice. We will talk more about Fibonacci numbers in the upcoming lessons.

Here is a recursion tree for binary recursion:



Binary recursion

Binary recursion: A type of Non-linear recursion

Alright, now that we are done with definitions, we will be deconstructing recursion in the next lesson.