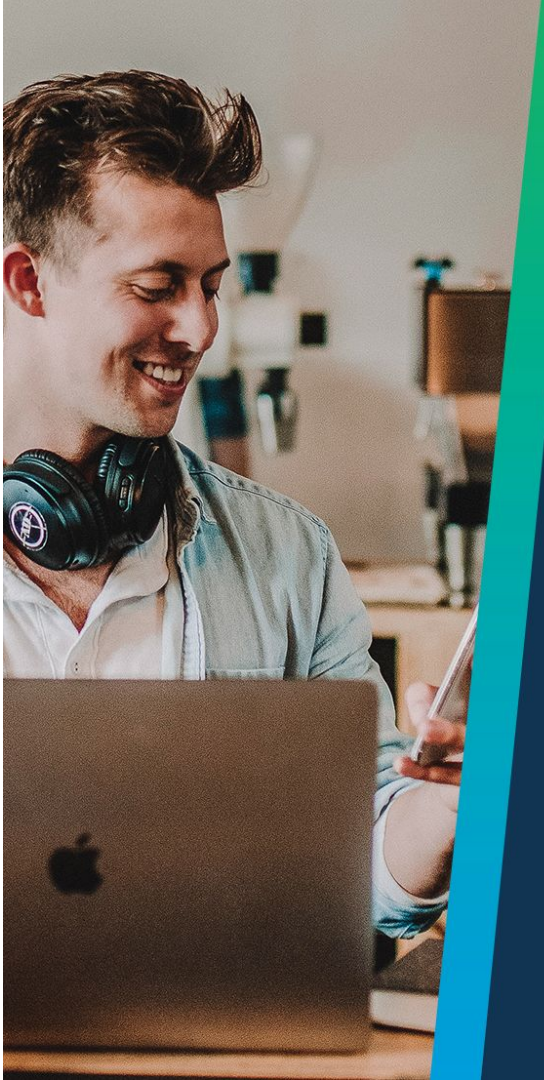


All Bootcamps

Hyperiondev

# Recursion

# Workshop – Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- No question is daft or silly - **ask them!**
- There are Q/A sessions midway and at the end of the session, should you wish to ask any follow-up questions.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support
- Report a safeguarding incident: http://hyperiondev.com/safeguardreporting
- We would love your feedback on lectures: https://hyperionde.wufoo.com/forms/zsgv4m40ui4i0g/

# Objectives

1. Recursion
    a. What is recursion?
    b. How to use recursion?

# What is Recursion?

Recursion is defined as a problem that is defined in terms of itself.

We face complex problems using recursion to break the problem down into simpler ones.

In simple terms, recursion is when a function calls itself.

# Recursive Function

Normally a recursive function uses conditional statements to determine whether or not to call the function recursively. The main benefits of using recursion are compactness of code, understandability of code, and having fewer variables.

Recursion and iteration (loops) can be used to achieve the same results. However, unlike loops, which work by explicitly specifying a repetition structure, recursion uses continuous function calls to achieve repetition.

# Recursive Function

Recursion is a somewhat advanced topic and problems that can be solved with recursion can also most likely be solved by using simpler looping structures. However, recursion is a useful programming technique that, in some cases, can enable you to develop natural, straightforward, simple solutions to otherwise difficult problems.

The following guidelines will help you to decide which method to use depending on a given situation:

- When to use recursion?
  - When compact, understandable, and intuitive code is required.
- When to use iteration?
  - When there is limited memory and faster processing is required.

# Recursive Function

There are two main requirements for a recursive function:

- Base case:
  - The function returns a value when a certain condition is satisfied, without any other recursive calls.
- Recursive call:
  - The function calls itself with an input that is a step closer to the base case.

Without these two main requirements we run the risk of creating an infinite recursive function, which will lead to a Stack Overflow.

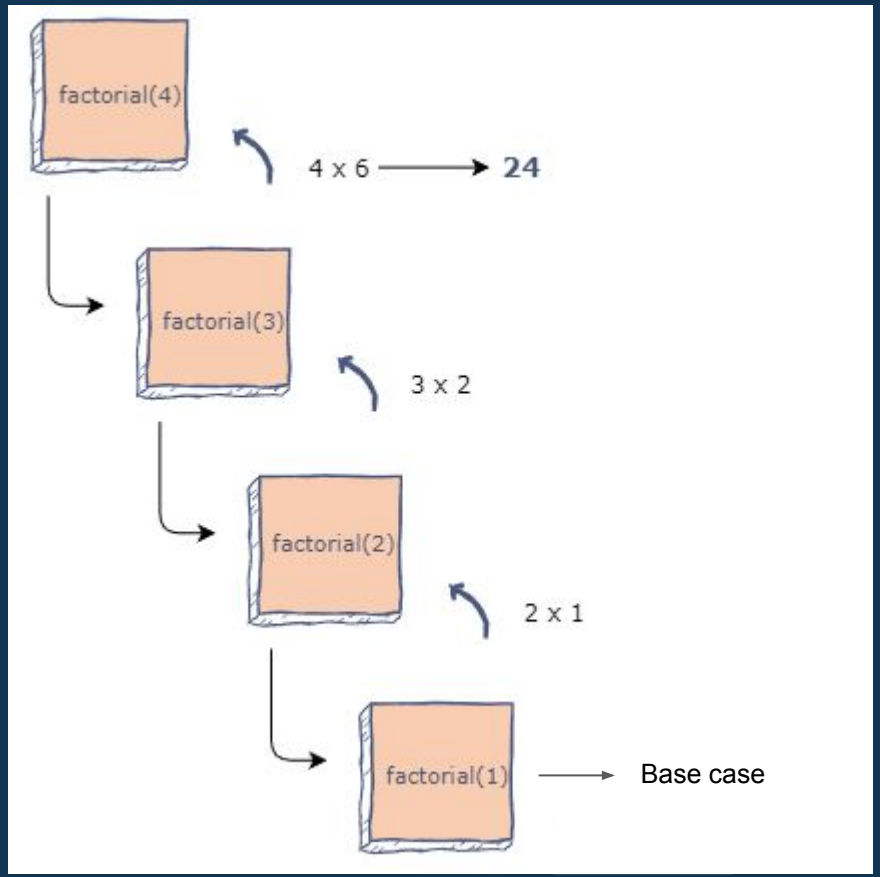
# Computing Factorials Using Recursion

As mentioned previously, a recursive function is a function that calls itself.

Many mathematical functions can be defined using recursion. A simple example is a factorial function. The factorial function, n! describes the operation of multiplying a number by all positive integers less than or equal to itself (excluding zero).

*For example: 4! = 4*3*2*1*

# Computing Factorials



# Computing Factorials

```
def factorial(num):
    if num == 1:
        return 1
    else:
        return num * factorial(num-1)
```

Hyperiondev

# Q & A Section

**Please use this time to ask any questions relating to the topic, should you have any.**

Hyperiondev

# Thank you for joining us

Stay hydrated
Avoid prolonged screen time
Take regular breaks
Have fun :)