Understanding a Recursive Problem

In this lesson, we will go over methods to help you visualize a recursive function.

We'll cover the following

Understanding the Problem

In the previous lessons, we learned the basic concept of recursion and its uses. Now, we will discuss how recursion works.

Let’s take a look at an example code:

```
def printPattern(targetNumber) :

  if (targetNumber <= 0) :
    print(targetNumber)
    return

  print(targetNumber)
  printPattern(targetNumber - 5)
  print(targetNumber)

# Driver Program
n = 10
printPattern(n)
```

On first glance, we notice that the `targetNumber` is decreased by 5 and `printPattern()` is being called again. However, there are two `print()` statements preceding and succeeding the recursive call.

Code Explanation

We want to print a pattern: $10\ 5\ 0\ 5\ 10$. Notice, we first decrease the `targetNumber` by 5 then increase the `targetNumber` by 5. However, the middle number (0) is printed only once.

Therefor, this becomes our **base case**:

```
if targetNumber <= 0:
  print targetNumber
  return
```

Now, the remaining numbers are printed twice on each side of the base case.

```
print targetNumber
printPattern(targetNumber - 5)
print targetNumber
```

This is our recursive case.

There are three methods commonly used to process the code flow of a recursive program and to **dry run** the code:

Visualizing Through a Stack

The concept of a stack is critical in recursion. The concept of recursive calls and their outputs are easier to understand when you visualize your function calls through a stack.

Let’s revisit this concept with an example:

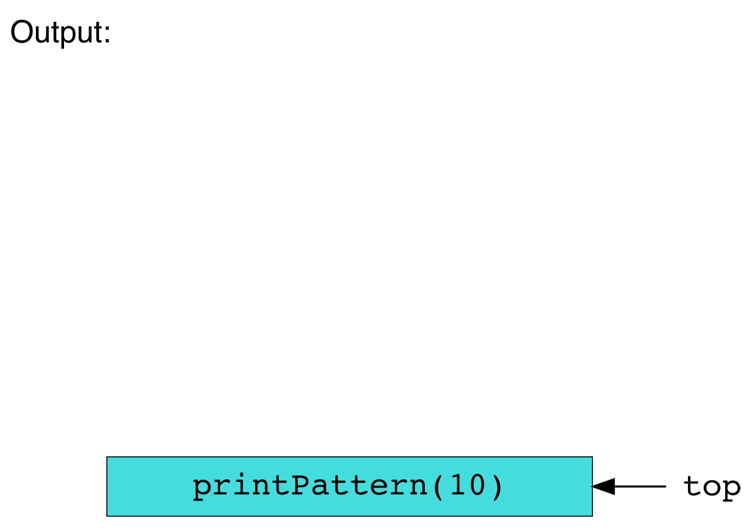
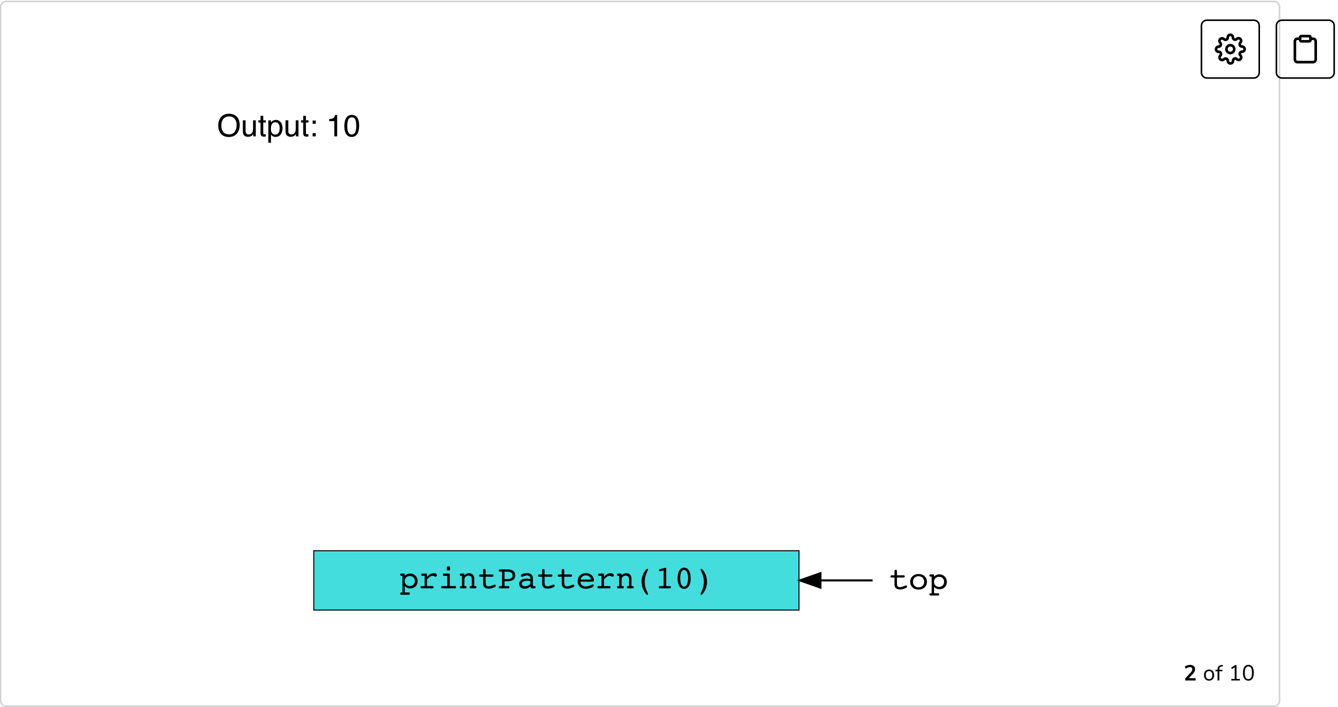


Illustration of how to visualize recursion through stack

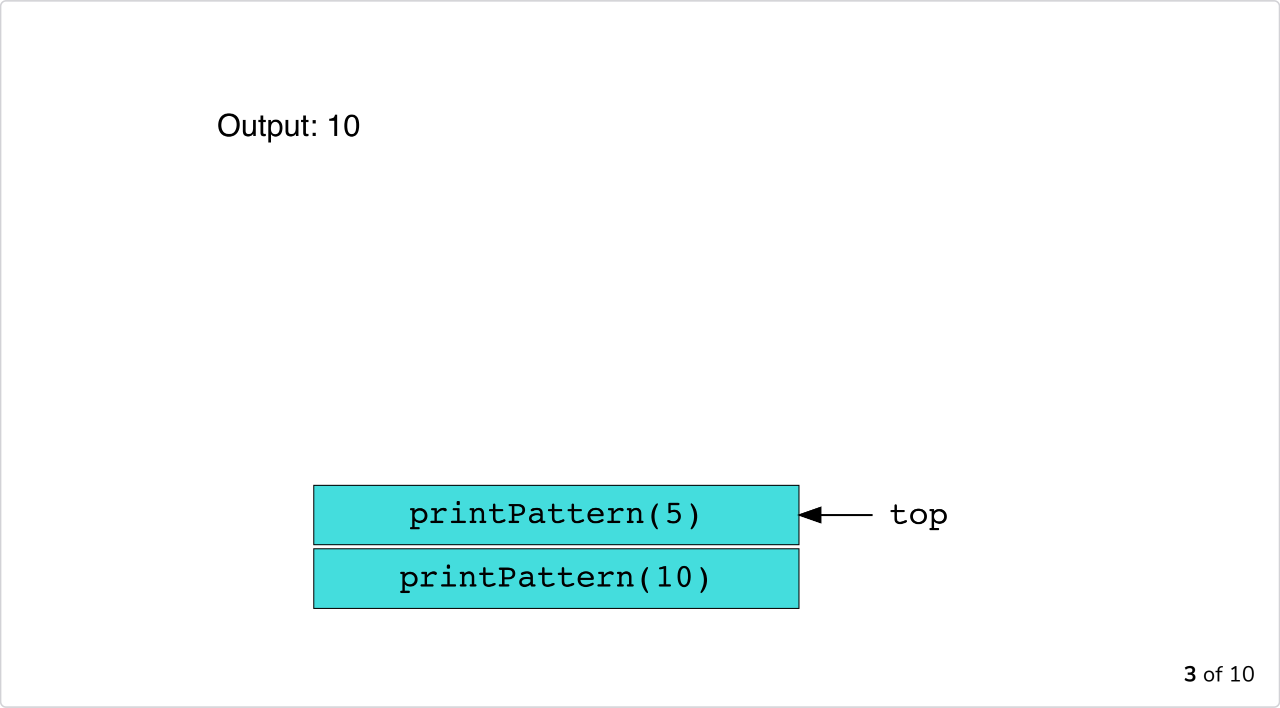
1 of 10



Output: 10

printPattern(10) ← top

2 of 10

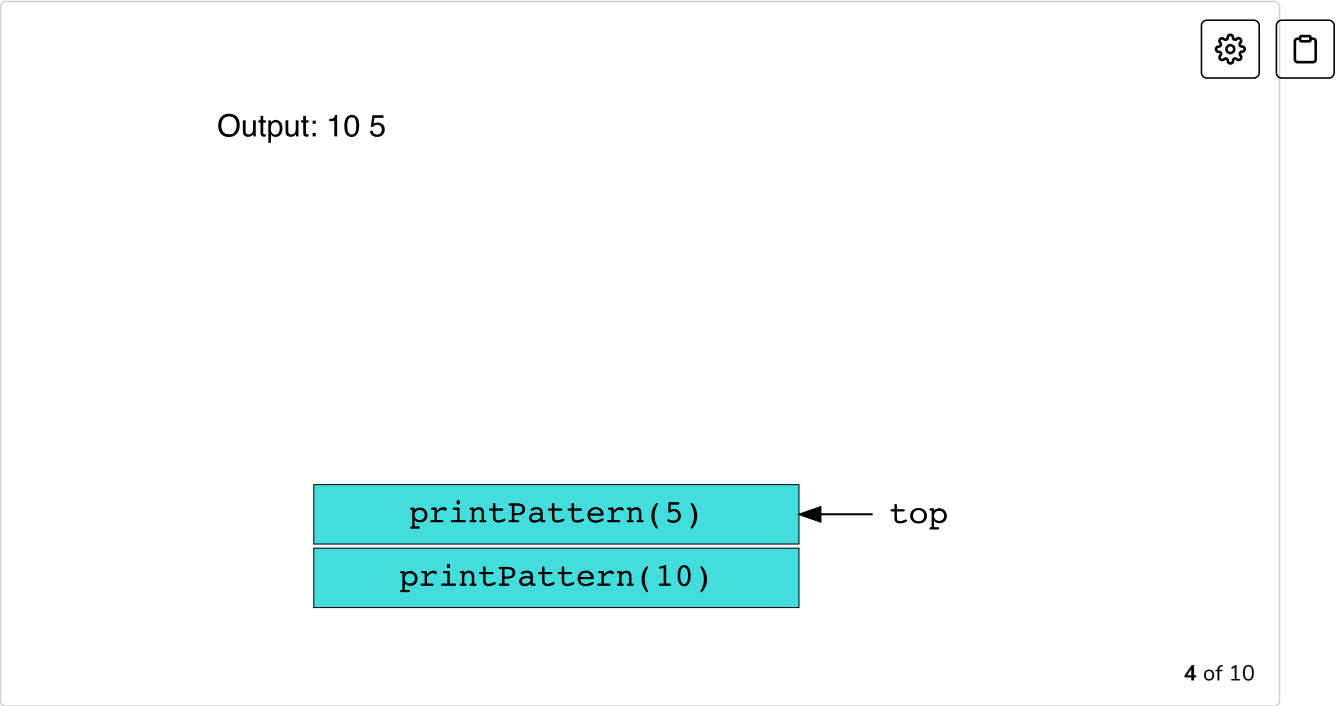


Output: 10

printPattern(5) ← top

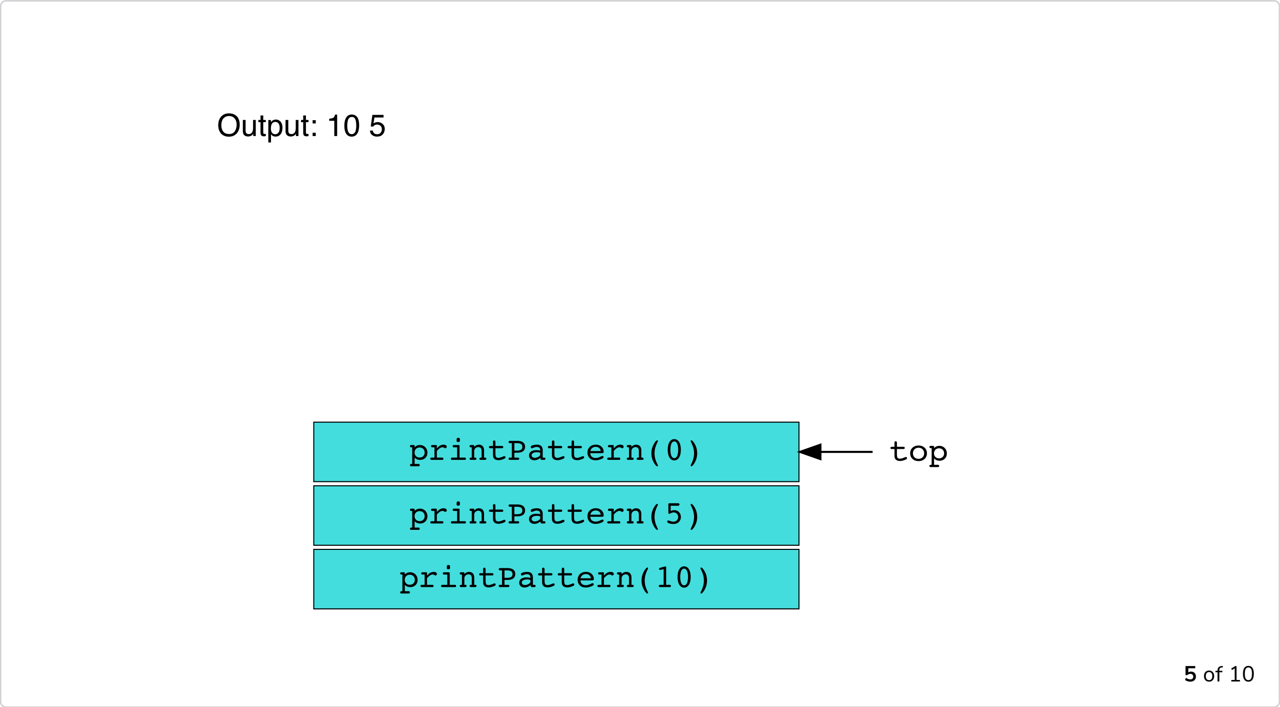
printPattern(10)

3 of 10



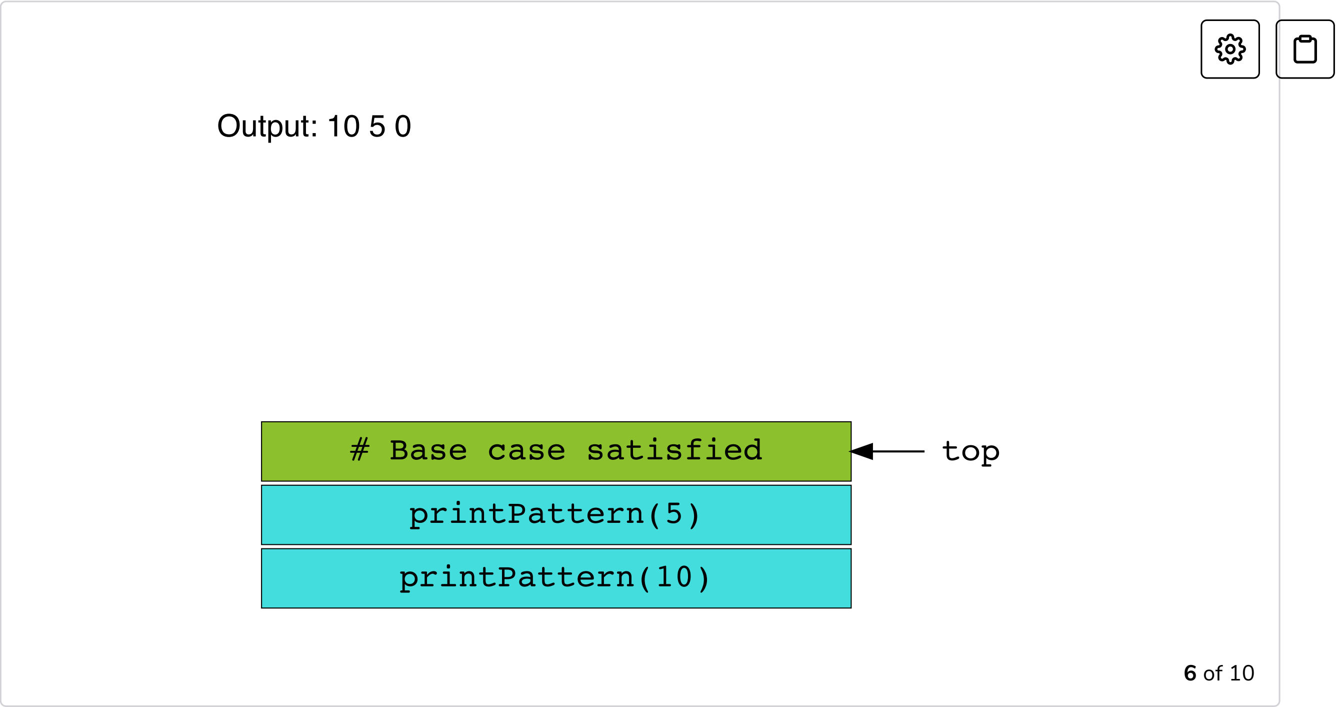
Output: 10 5

4 of 10



Output: 10 5

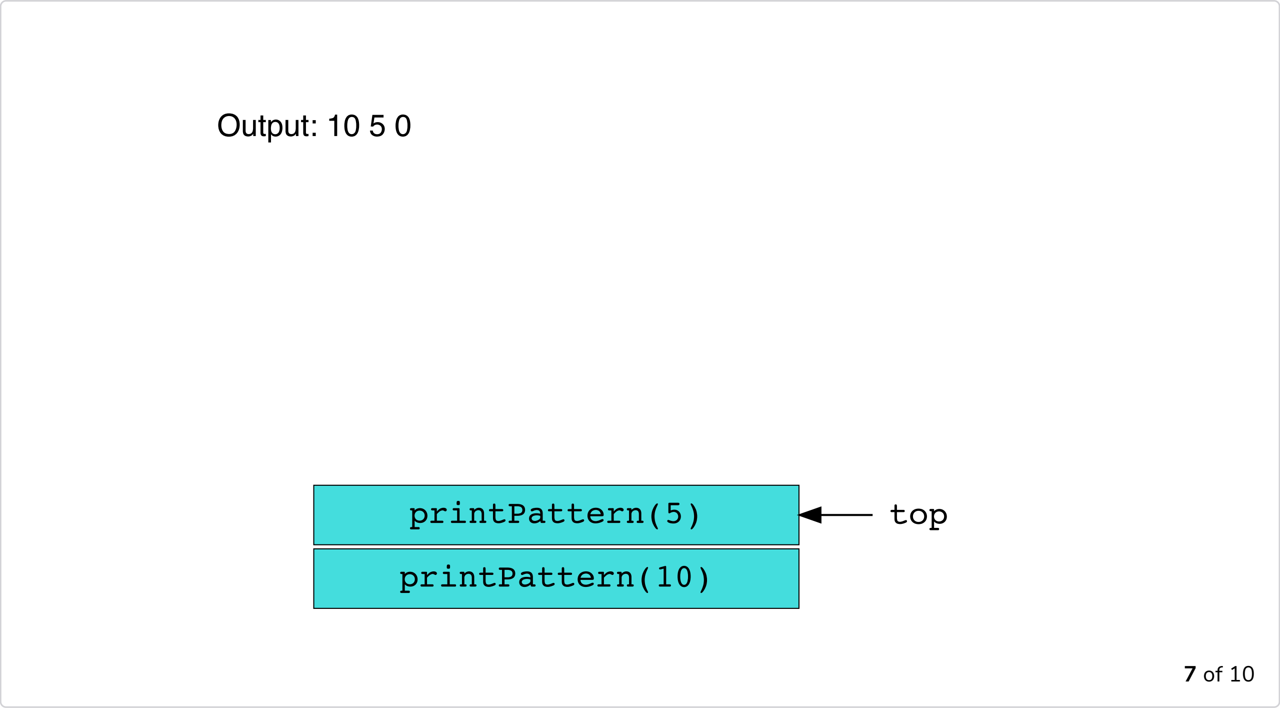
5 of 10



Output: 10 5 0

```
# Base case satisfied   ← top
printPattern(5)
printPattern(10)
```

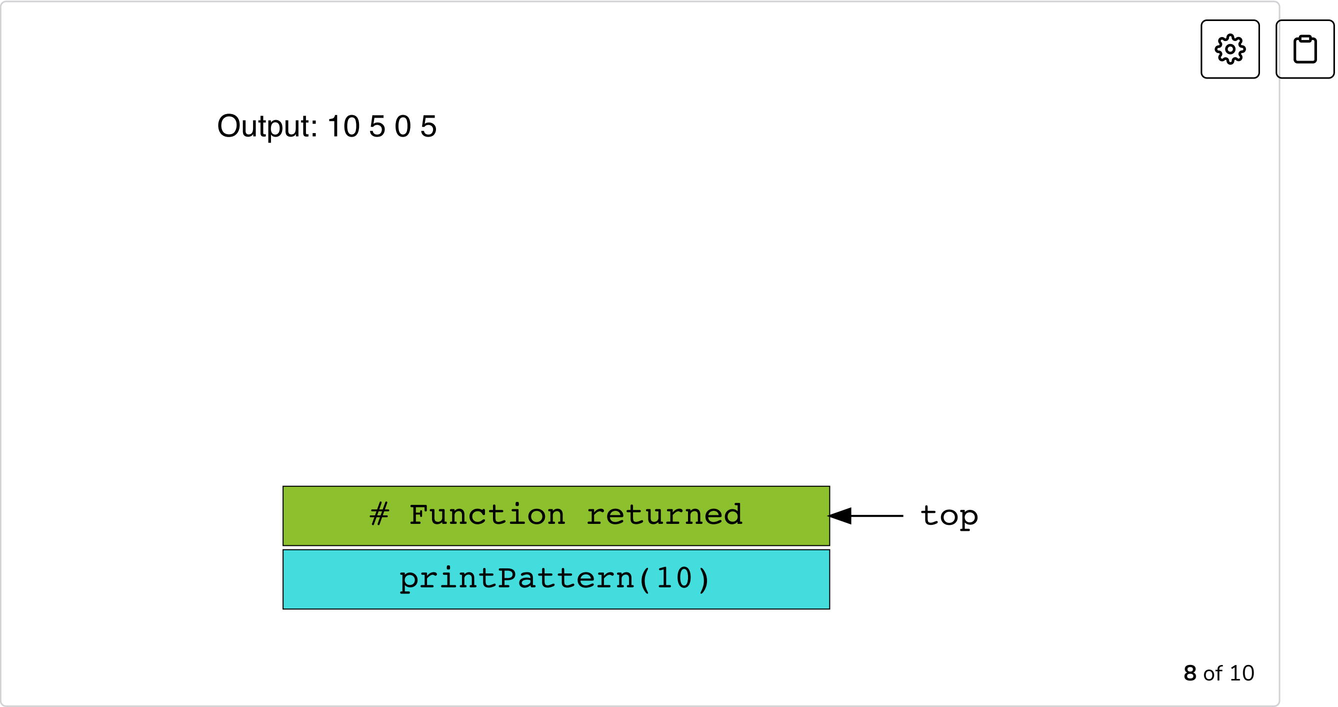
6 of 10



Output: 10 5 0

```
printPattern(5)   ← top
printPattern(10)
```

7 of 10



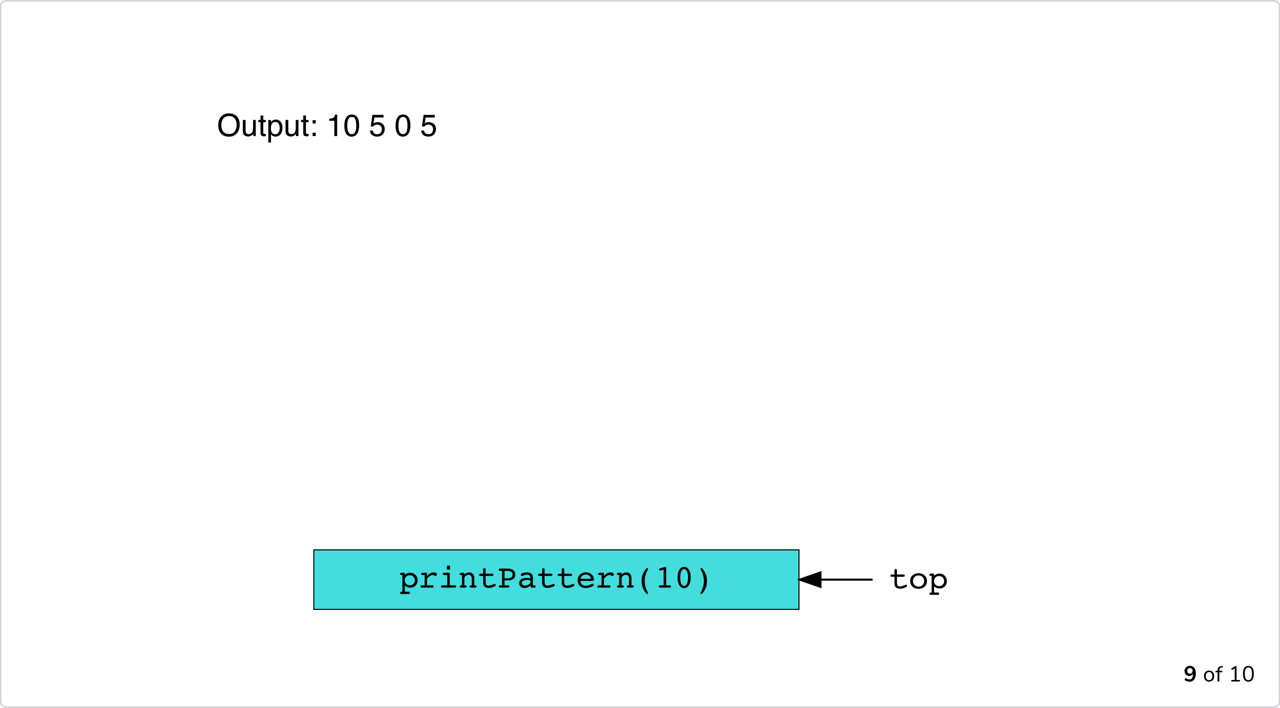
Output: 10 5 0 5

# Function returned

printPattern(10)

top

8 of 10

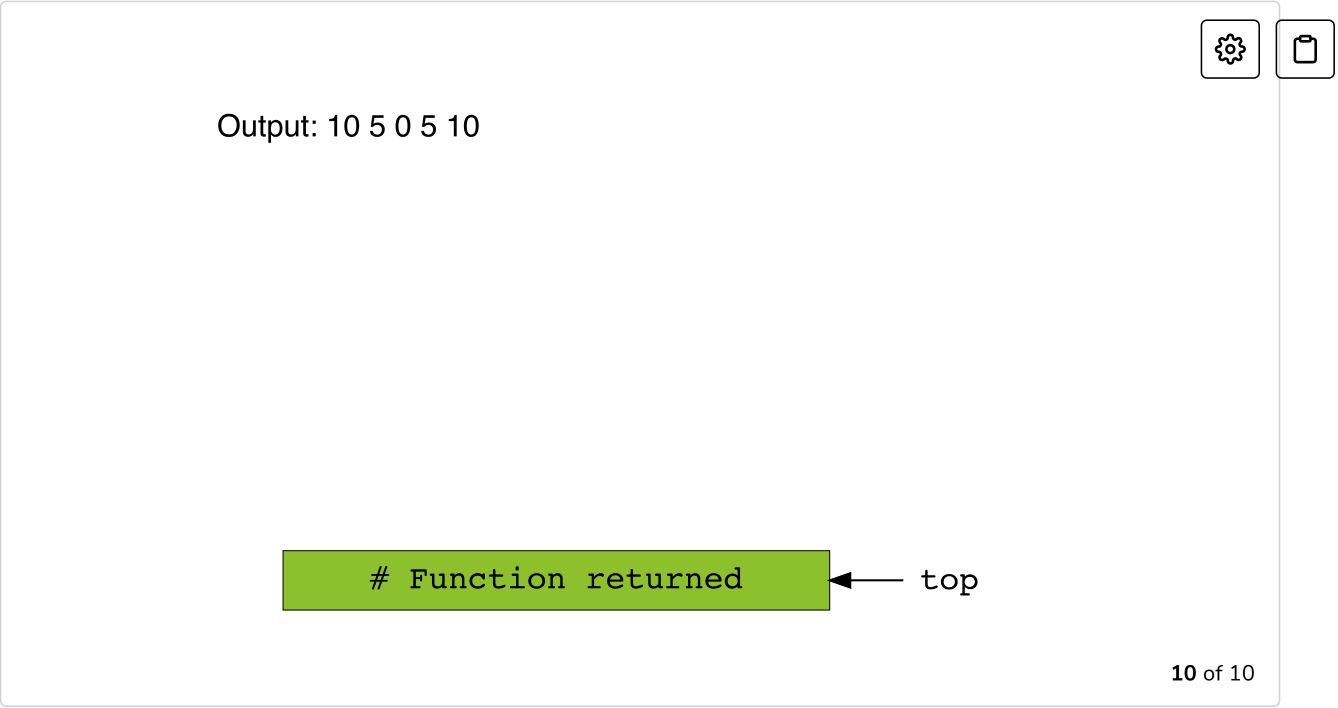


Output: 10 5 0 5

printPattern(10)

top

9 of 10



Output: 10 5 0 5 10

Drawing a Recursive Tree

Recursive functions usually act like a tree. The parent is the main function call and each recursive call becomes a child node.

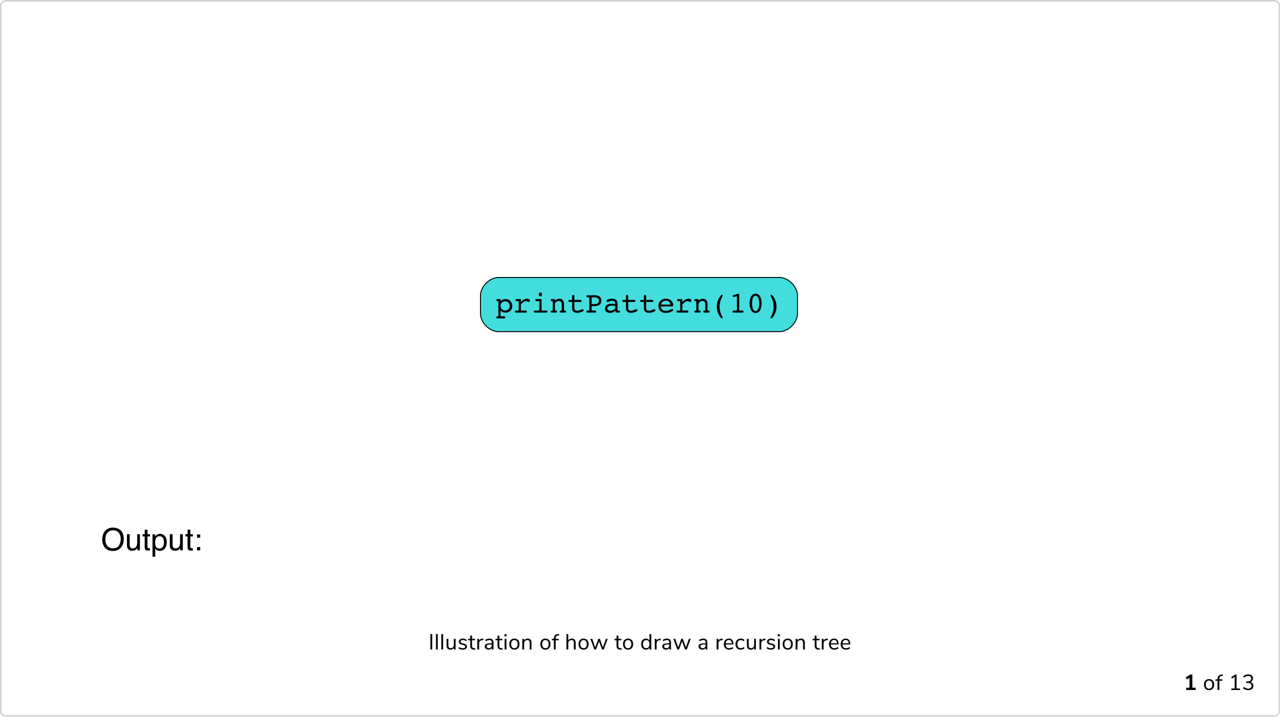


Illustration of how to draw a recursion tree

> We will be using the **recursive tree** method for representing recursive function calls throughout this course.

Keeping a Track of Your Variables

Keeping track of variables can help dry run complicated codes. It is a detailed method and can be time-consuming. However, this method is helpful while writing recursive functions.

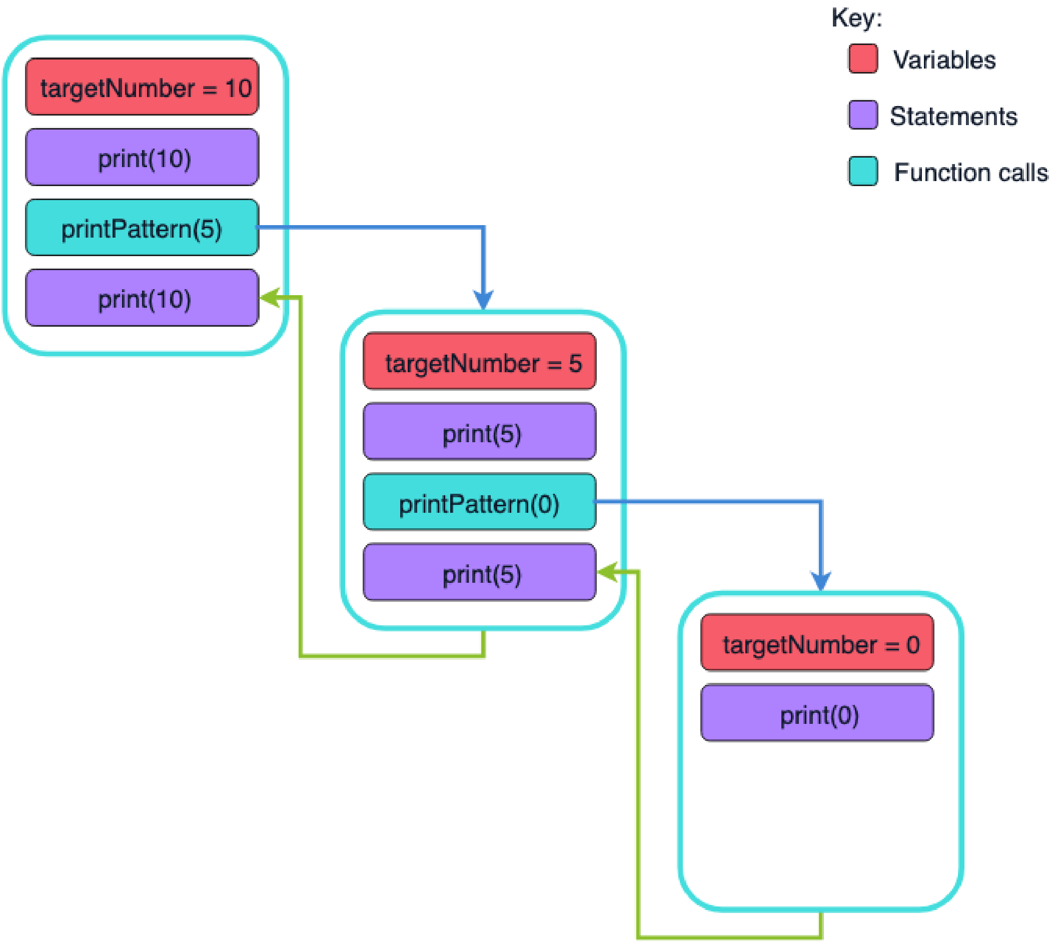


Illustration of how to track variables in recursion

In the next lesson, there will be a quick quiz for you to test your understanding of this chapter.

← Back

When to Use Recursion?

Next →

Test Your Knowledge 1

☑ Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/understanding-a-recursive-problem__recursion-fundamentals__recursion-for-coding-interviews-in-python)