RECURSION

Prof. K. Adisesha

BE, M.Sc., M.Th., NET, (Ph.D.)

Learning objectives

- Introduction
- Recursive Functions
- How Recursive Works
- Recursive in Python
- Recursive functions Examples
- Recursive Vs Iteration

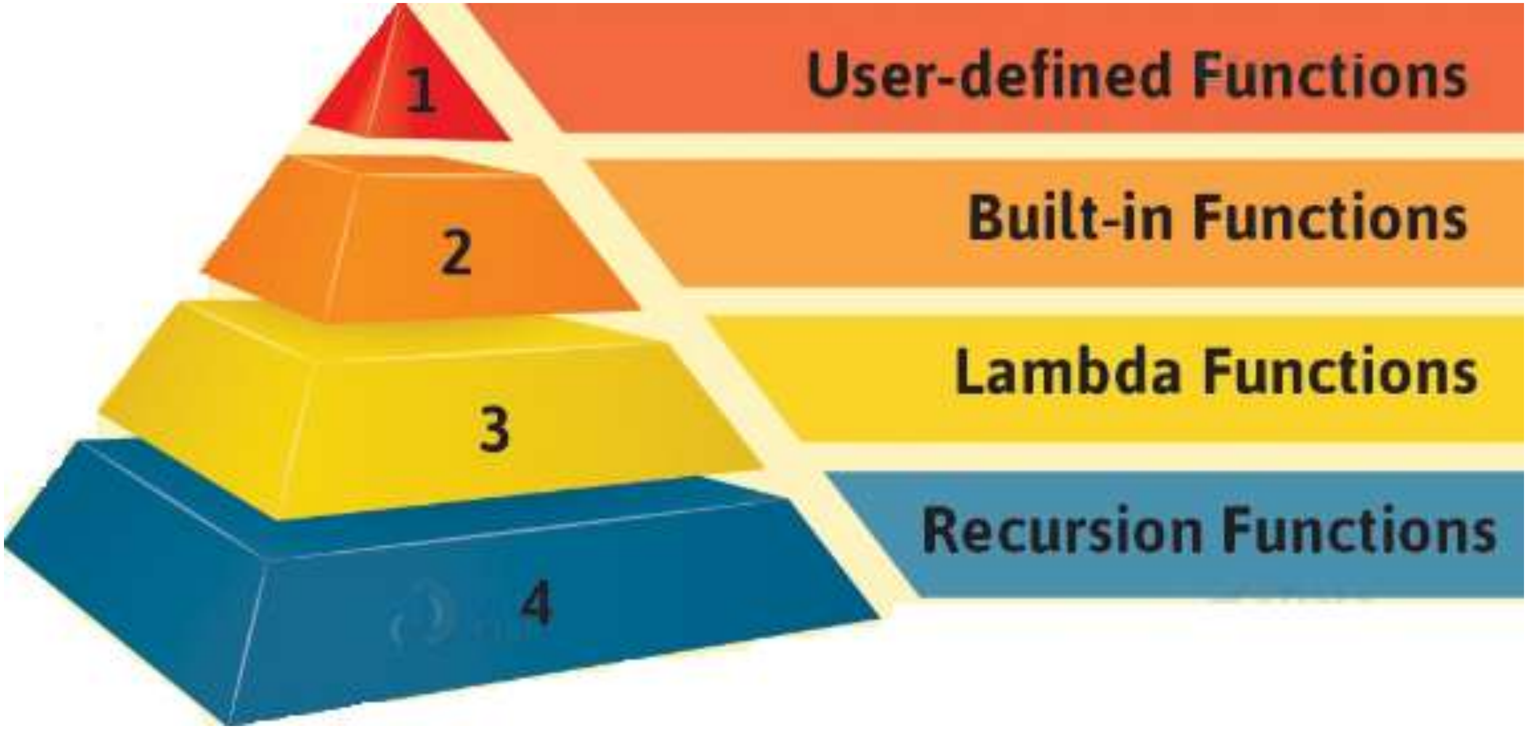
What Are Functions?

- A function is a block of code which only runs when it is called.
- Functions are sub-programs which perform tasks which may need to be repeated.
- Some functions are "bundled" in standard libraries which are part of any language's core package. We've already used many built-in functions, such as input(), eval(), etc.
- Functions are similar to methods, but may not be connected with objects
- Programmers can write their own functions

Types of Functions

Different types of functions in Python:

Python built-in functions, Python recursion function, Python lambda function, and Python user-defined functions with their syntax and examples.



Recursion

Recursion:

- A technique for solving a large computational problem by repeatedly applying the same procedure to reduce it to successively smaller problems.
- Recursion refers to a programming technique in which a function calls itself either directly or indirectly
- Recursion is a common mathematical and programming concept.
- A recursive procedure has two parts:
  - One or more base cases
  - A recursive steps.

Recursion

- This has the benefit of meaning that you can loop through data to reach a result.
- It means that a function calls itself.
- Recursion can be two types:
  - **Direct Recursion**
  - **Indirect Recursion**
- The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power.

Recursion

- **Direct Recursion**: if function calls itself directly from its function body.

  Example:

  ```
  def recur():
   recur()   # function recur() calling itself
  ```

- **Indirect Recursion**: if a function calls another function, which calls its caller function

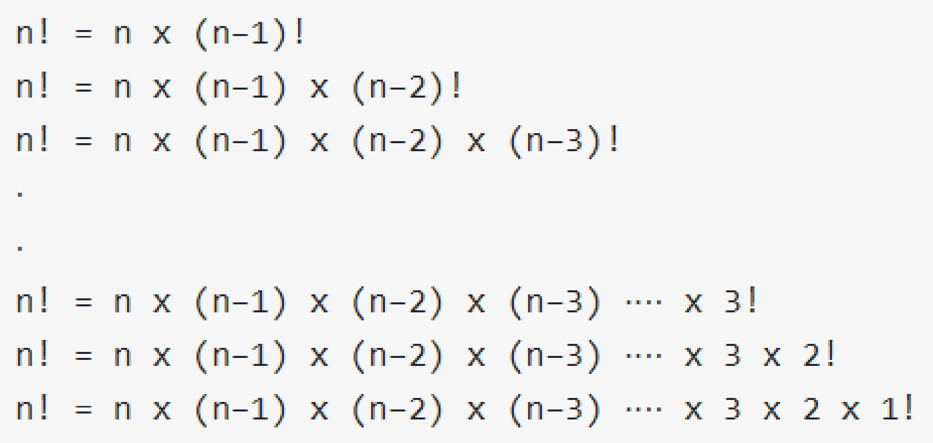
  Example:

  ```
  def recur-A():
   recur-B()      # function recur-A() calling recur-B(),
                         which calls recur-A()
  def recur-B():
   recur-A()
  ```

How Recursive Works

Overview of how recursive function works:

- Recursive function is called by some external code.
- If the base condition is met then the program do something meaningful and exits.
- Otherwise, function does some required processing and then call itself to continue recursion. Here is an example of recursive function used to calculate factorial.
- Example:
- Factorial is denoted by number followed by (!) sign i.e 4!
- Steps:
  - 4! = 4 * 3 * 2 * 1
  - 2! = 2 * 1
  - 0! = 1



```
n! = n x (n-1)!
n! = n x (n-1) x (n-2)!
n! = n x (n-1) x (n-2) x (n-3)!
.
.
n! = n x (n-1) x (n-2) x (n-3) ···· x 3!
n! = n x (n-1) x (n-2) x (n-3) ···· x 3 x 2!
n! = n x (n-1) x (n-2) x (n-3) ···· x 3 x 2 x 1!
```

How Recursive Works

- However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.
- Sensible Recursive code is the one that fulfills following requirements :
  - It must have a case, whose result is known or computed without any recursive calling -The BASE CASE.
  - The BASE CASE must be reachable for some argument/parameter.
  - it also have Recursive Case, where by function calls itself.
- Example:

```
def factorial_recursive(n):
    # Base case: 1! = 1
    if n == 1:
        return 1
    # Recursive case: n! = n * (n-1)!
    else:
        return n * factorial_recursive(n-1)
print("\n\n Recursion Example Results")
factorial_recursive(6)
```

Recursive in Python

Writing a Recursive Function.

- Before you start working recursive functions, you must know that every recursive function must have at least two cases :
  - The **Recursive Case** (or the inductive case)
  - The **Base Case** (or the stopping case)always required
- The Base Case in a recursive program must be reachable that causes the recursion to end.
- The Recursive Case is the more general case of the problem we're trying to solve using recursive call to same function.
- Example: function xn, the recursive case would be :

  Power (x, n) = x * Power (x, n - 1)

  The base cases would be:

  Power(x, n)=x when n=1

  Power(x, n)=1 when n=0

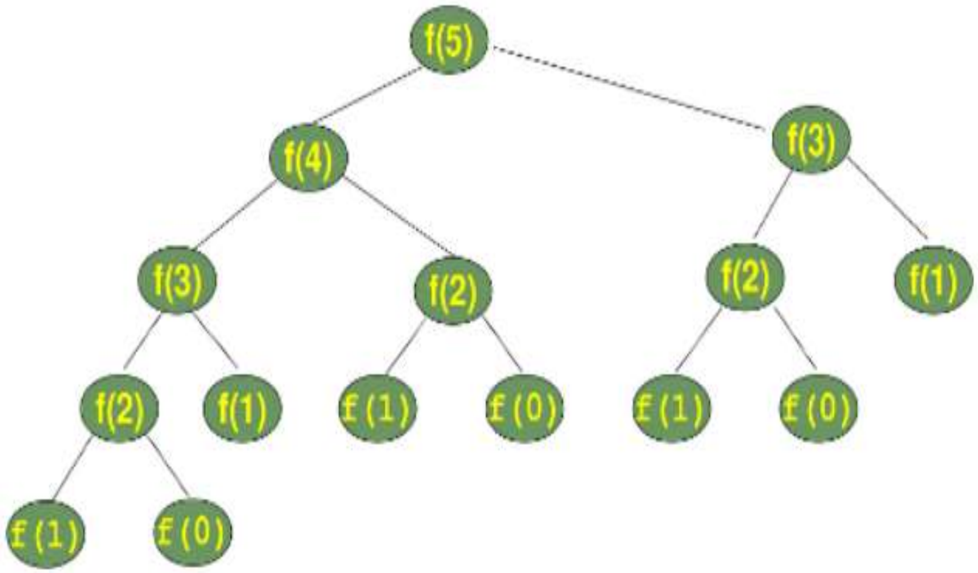
  Other cases(when n<0) ignoring simplicity sake

Recursive in Python

Writing a Recursive Function.

- The Fibonacci numbers are easy to write as a Python function.
- It's more or less a one to one mapping from the mathematical definition:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

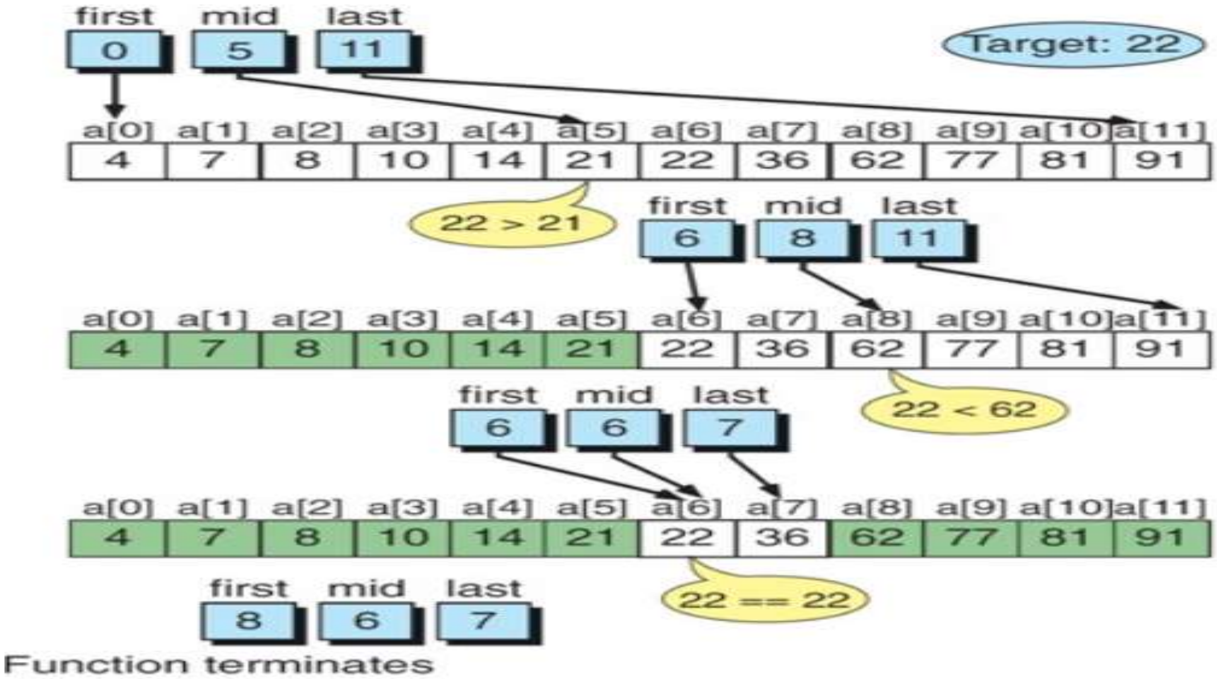


The order in which the functions are called.
fib() is substituted by fib().

Binary Search

Binary Search Techniques.

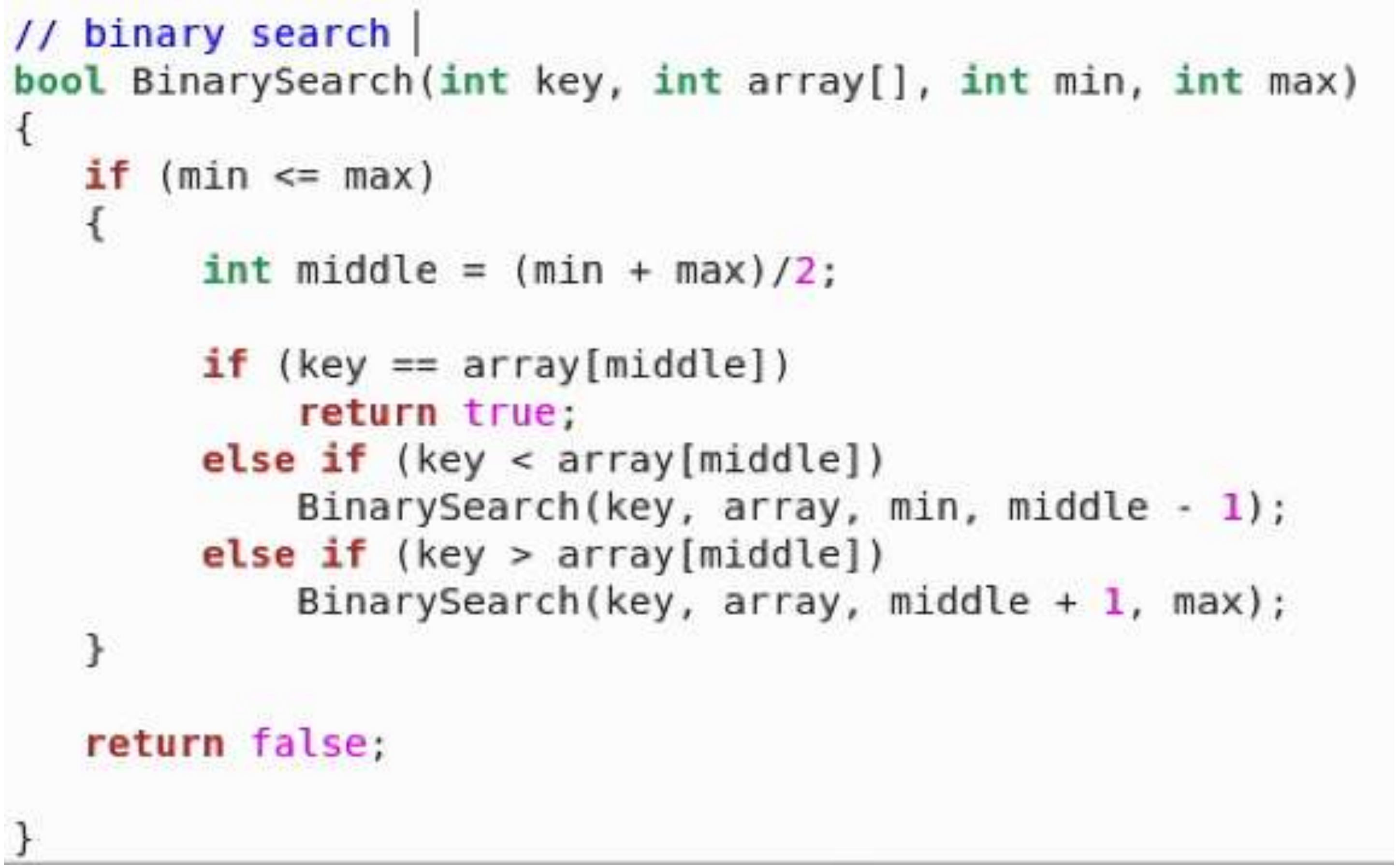
- Popular algorithm that used recursion successfully is binary search algorithm.
- Binary search works for only sorted array whereas linear search work for both sorted as well as unsorted array.
- The process of binary search is illustrated in the figure:



Binary Search

Binary Search Algorithm.

- Popular algorithm that used recursion successfully is binary search algorithm.

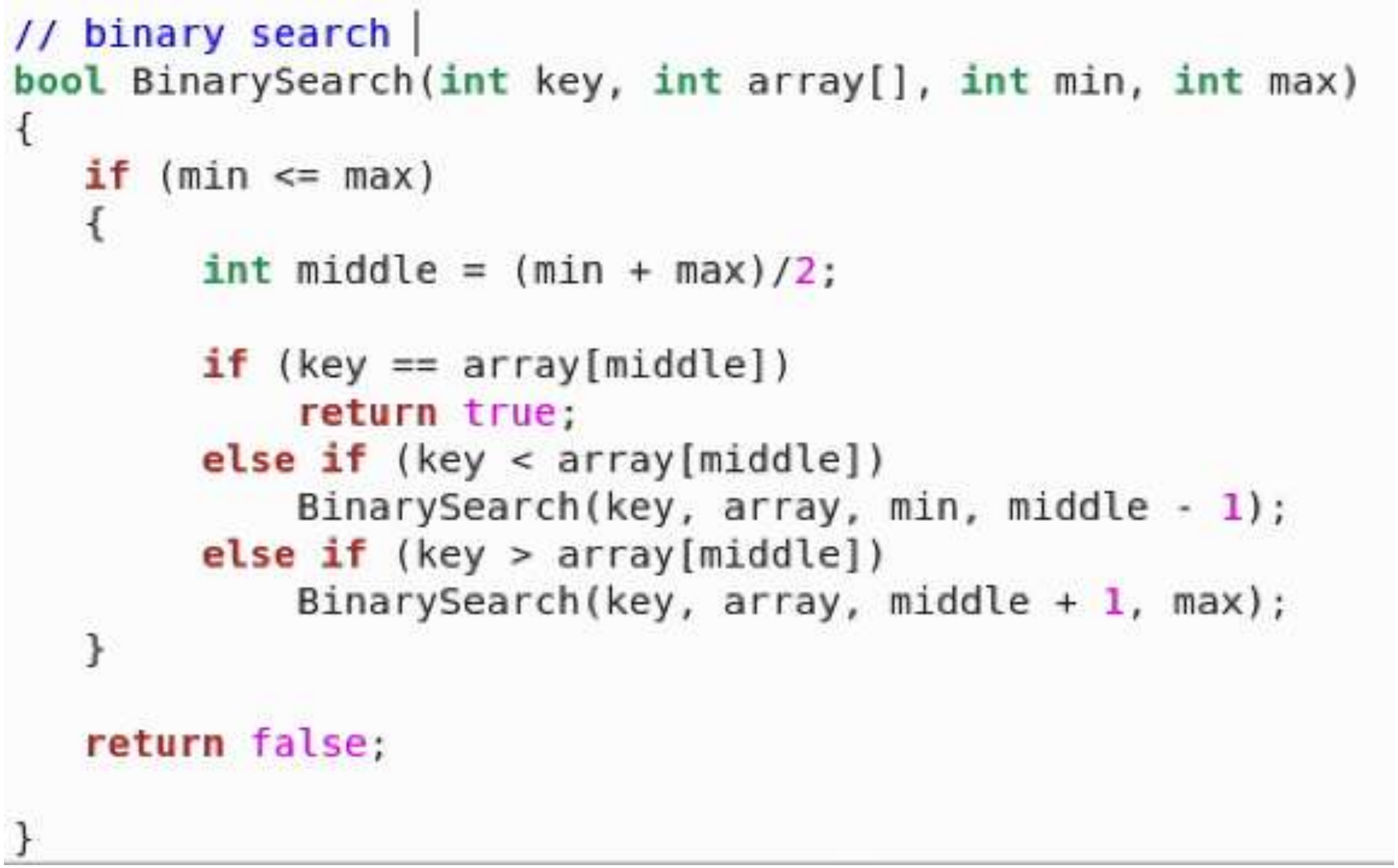


```
// binary search
bool BinarySearch(int key, int array[], int min, int max)
{
    if (min <= max)
    {
        int middle = (min + max)/2;

        if (key == array[middle])
            return true;
        else if (key < array[middle])
            BinarySearch(key, array, min, middle - 1);
        else if (key > array[middle])
            BinarySearch(key, array, middle + 1, max);
    }

    return false;

}
```

Binary Search

Binary Search Algorithm.

- Popular algorithm that used recursion successfully is binary search algorithm.



```
// binary search
bool BinarySearch(int key, int array[], int min, int max)
{
    if (min <= max)
    {
        int middle = (min + max)/2;

        if (key == array[middle])
            return true;
        else if (key < array[middle])
            BinarySearch(key, array, min, middle - 1);
        else if (key > array[middle])
            BinarySearch(key, array, middle + 1, max);
    }

    return false;

}
```

Recursion vs. Iteration

Difference between Recursion and Iteration

- A program is called recursive when an entity calls itself.
- A program is call iterative when there is a loop (or repetition).

PROPERTY	RECURSION	ITERATION
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high(generally exponential) time complexity.	Relatively lower time complexity (generally polynomial-logarithmic).

Conclusion!

- We learned about the Python function.
- Recursive Functions
- How Recursive Works
- Recursive in Python
- Recursive functions Examples
- Recursive Vs. Iteration

Thank you