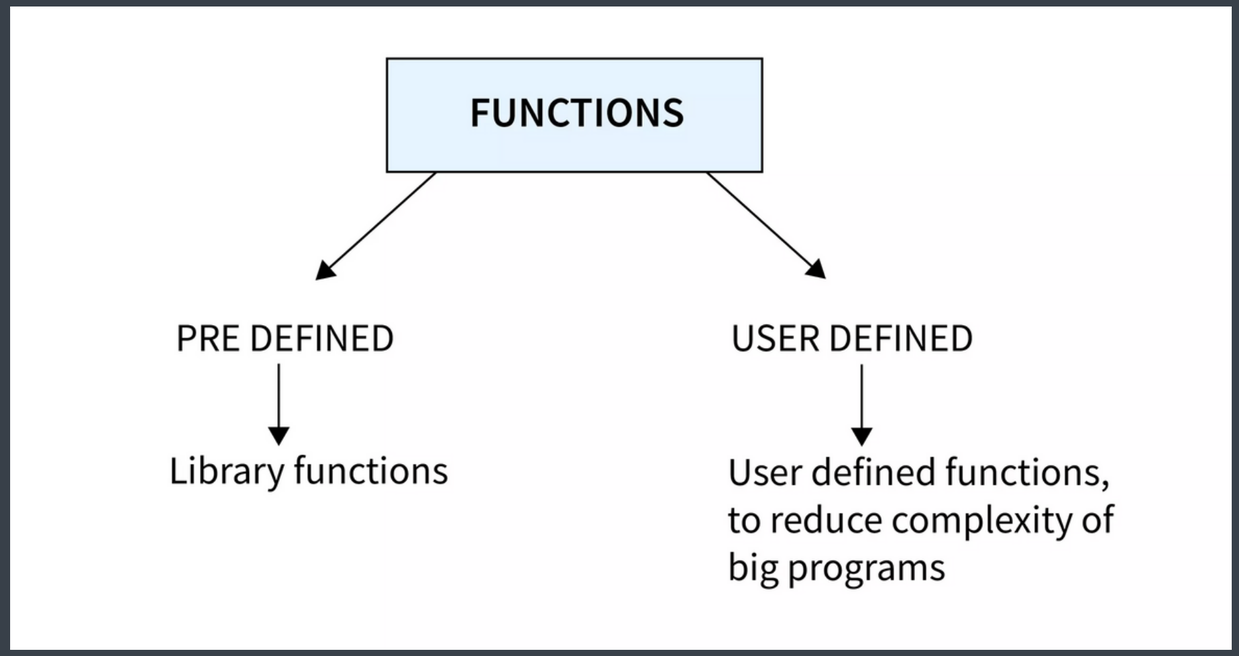
Functions

Function

Functions are blocks of organised, reusable code that perform a specific task

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Types


FUNCTIONS
PRE DEFINED
Library functions
USER DEFINED
User defined functions, to reduce complexity of big programs

Declaration/Defining

You define a function using the def keyword, followed by the function name, parentheses (), and a colon :

Syntax

```
def function_name(parameters):
        # Function body
```

Function Call

To execute a function and run the code inside it, you call the function by using its name followed by parentheses ().

Syntax

```
function_name(arguments)
```

Parameters and Arguments

Functions can take input values known as parameters or arguments.
Parameters are defined in the function's parentheses during the function definition.
Arguments are the actual values passed to the function when it is called.

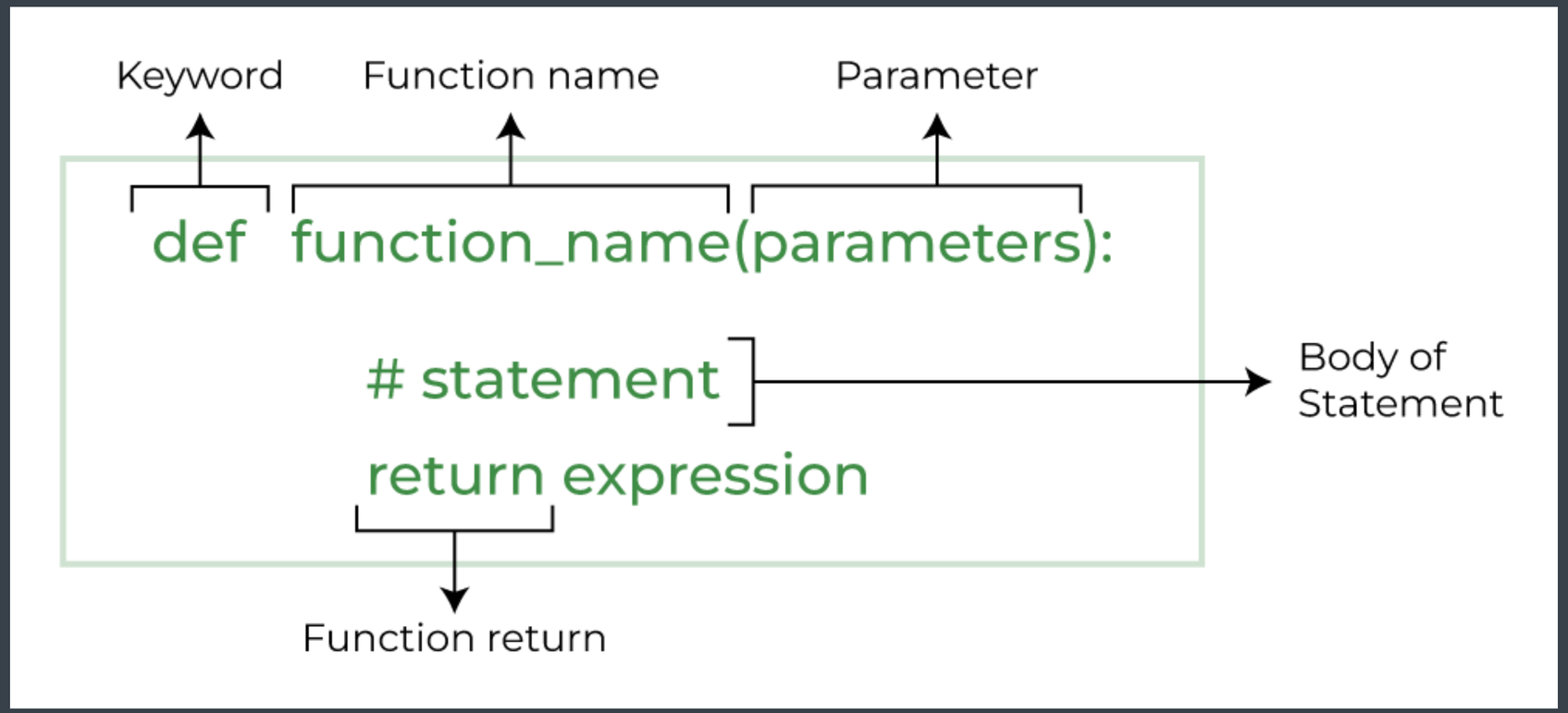
Return Statement

Functions can return a value using the return statement. The returned value can be assigned to a variable or used in expressions.

```
def add(a, b):
    return a + b

result = add(3, 5)
print(result)  # Output: 8
```

All at One Place



Positional Arguments

These are the most common type of arguments and are passed to a function based on their position. The order of the arguments in the function call must match the order of the parameters in the function definition.

```
def add_numbers(a, b):
    return a + b

result = add_numbers(5, 3)
print(result)  # Output: 8
```

Keyword Arguments

You can pass arguments to a function using their names explicitly, known as keyword arguments. This allows you to pass arguments in any order.

```
def personal_info(name, age):
    print("Name:", name)
    print("Age:", age)

personal_info(age=30, name="John")
```

```
Name: John
Age: 30
```

Default Arguments

Default arguments are used to provide default values for parameters in case no value is passed during the function call. If no argument is provided for a default parameter, the default value will be used.

```
def greet_user(name, greeting="Hello"):
    return greeting + ", " + name + "!"

greeting1 = greet_user("Bob")
greeting2 = greet_user("Charlie", "Hi")

print(greeting1)  # Output: Hello, Bob!
print(greeting2)  # Output: Hi, Charlie!
```

Scope of Variables

Global Scope

Variables defined outside of any function or block have a global scope. They can be accessed from anywhere in the program, including inside functions.

```
global_var = 10

def my_function():
    print(global_var)  # Accessible (prints 10)

print(global_var)  # Accessible (prints 10)
```

Local Scope

Variables defined inside a function have a local scope. They are accessible only within the function where they are defined and not from outside the function.

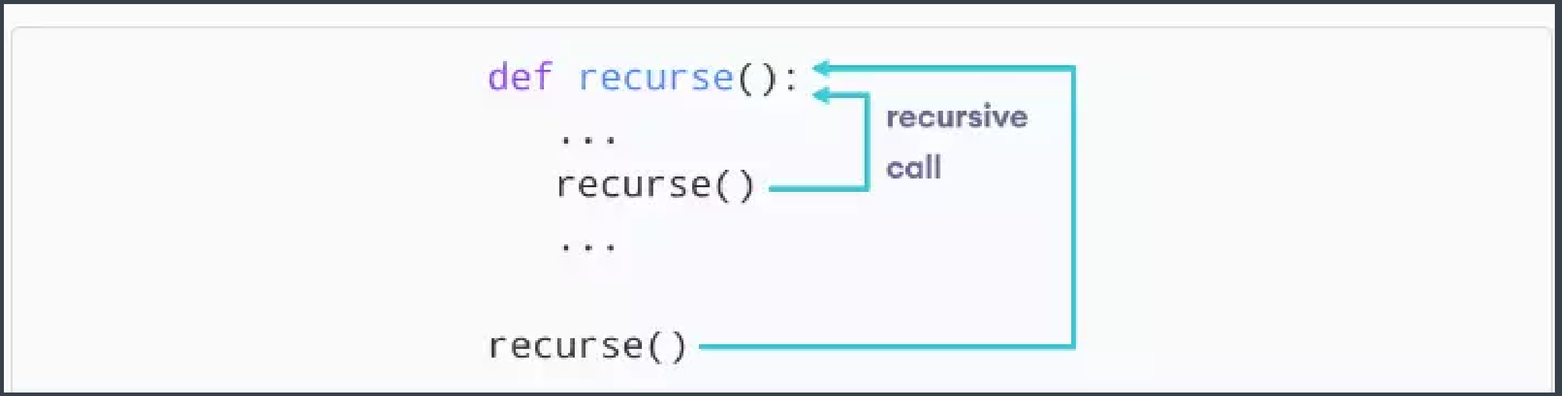
```
def my_function():
    local_var = 5
    print(local_var)  # Accessible (prints 5)

my_function()
# print(local_var)  # Not accessible (raises NameError)
```

Recursion

Recusive function

A function that calls itself within its own definition to solve a problem or perform a task

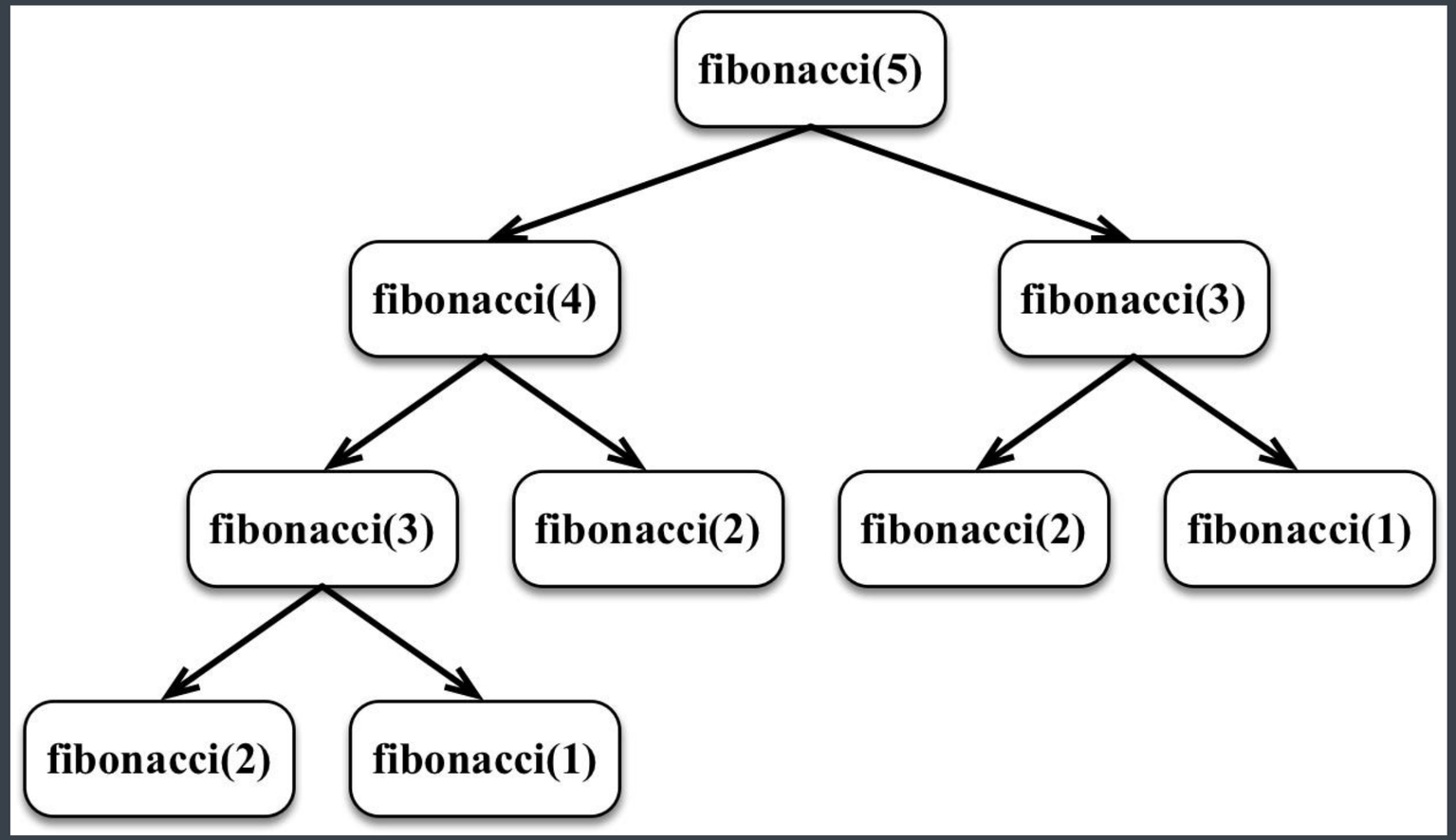


Recursive function

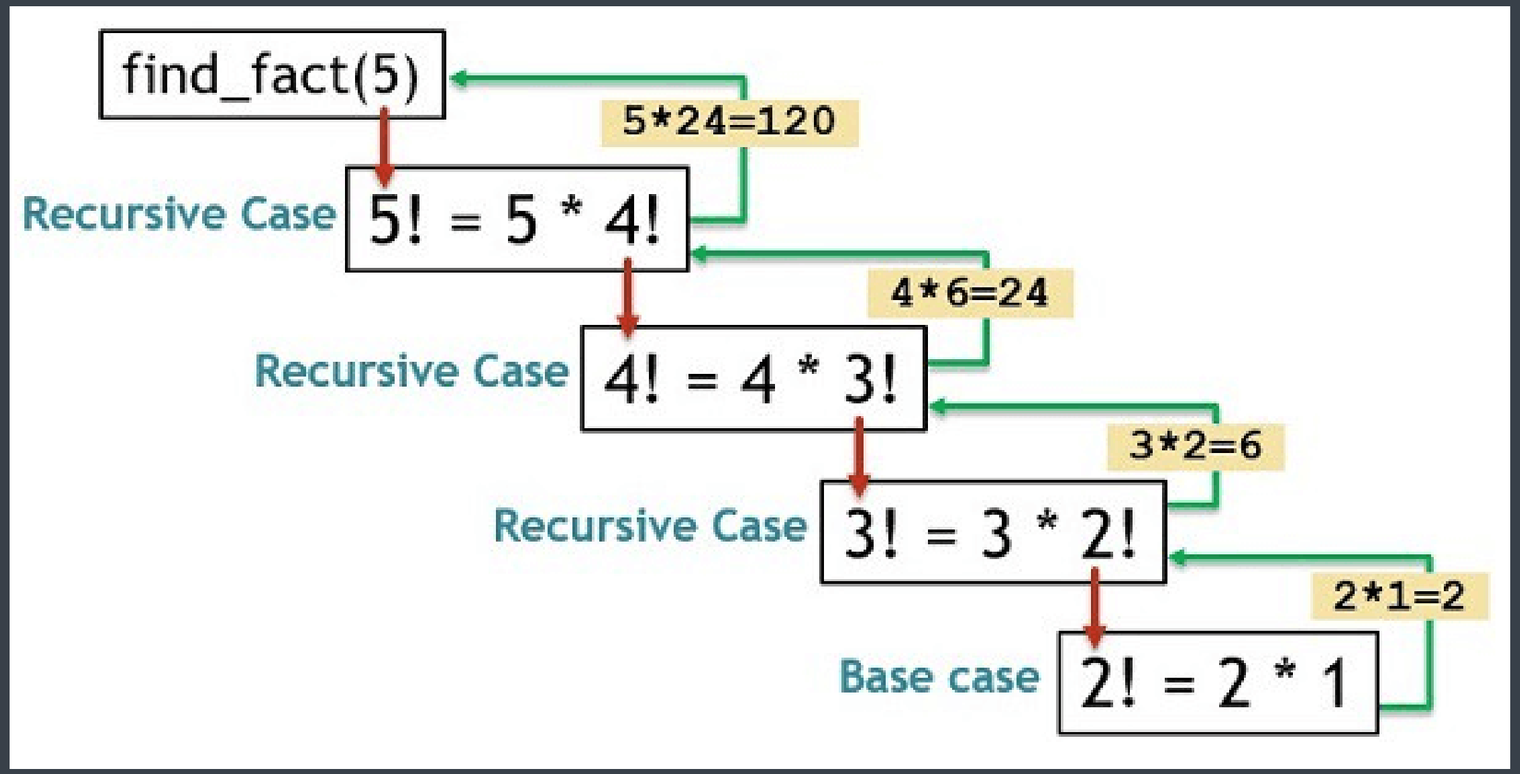
Base Case: The base case is the condition that specifies when the function should stop calling itself and return a result directly. It acts as the stopping criterion for the recursion, preventing infinite recursion.

Recursive Case: The recursive case is the part of the function that calls itself with a modified version of the input, leading to smaller instances of the same problem.

Recusive function for fibonacci



Recusive function for factorial



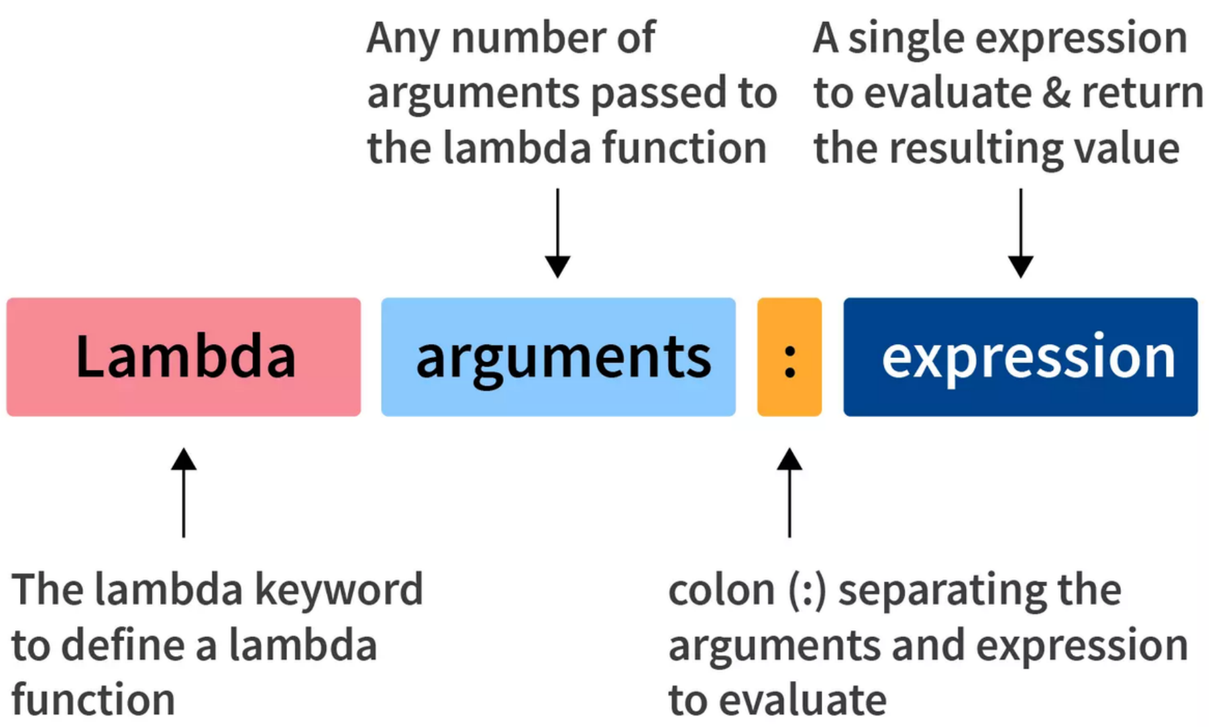
Lambda

Lambda function

Is a small, one-line function that can have any number of arguments but can only have one expression.

Syntax

lambda arguments: expression


Any number of arguments passed to the lambda function
A single expression to evaluate & return the resulting value
Lambda
arguments
:
expression
The lambda keyword to define a lambda function
colon (:) separating the arguments and expression to evaluate

Addition of Two Numbers:

```
add = lambda x, y: x + y
result = add(3, 5)
print(result)  # Output: 8
```

Squaring a Number:

```
square = lambda x: x ** 2
result = square(4)
print(result)  # Output: 16
```