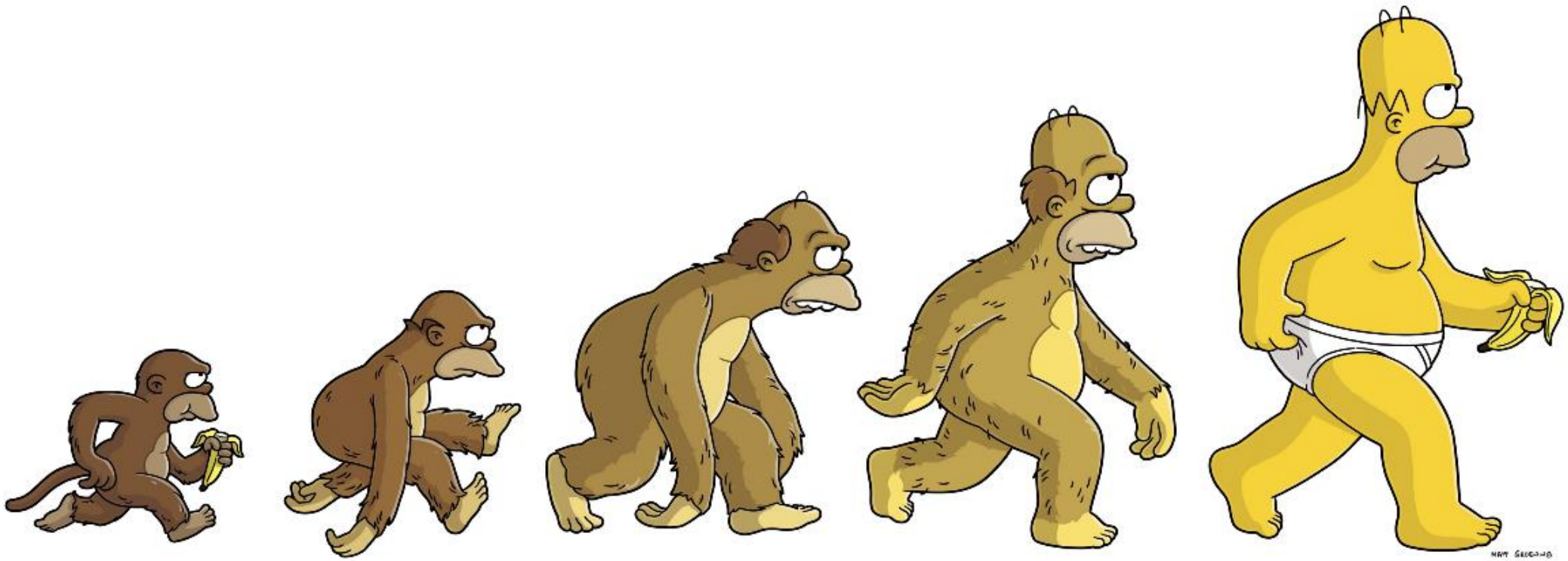
Functional Programming



Functional programming is a programming paradigm in which it is tried to bind each and everything in pure mathematical functions. It is a declarative type of programming style that focuses on what to solve rather than how to solve (aimed by the imperative style of programming).

It's Concepts

“Pure Functions”

Pure functions have two important properties, they:

- Always produce the same output with the same arguments disregard of other factors. This property is also known as immutability
- Are deterministic. Pure functions either give some output or modify any argument or global variables i.e. they have no side-effects

```python
def addition(x:int, y:int) -> int:
  return x+y
```

```python
def call_server():
    return requests.get('google.com')
```

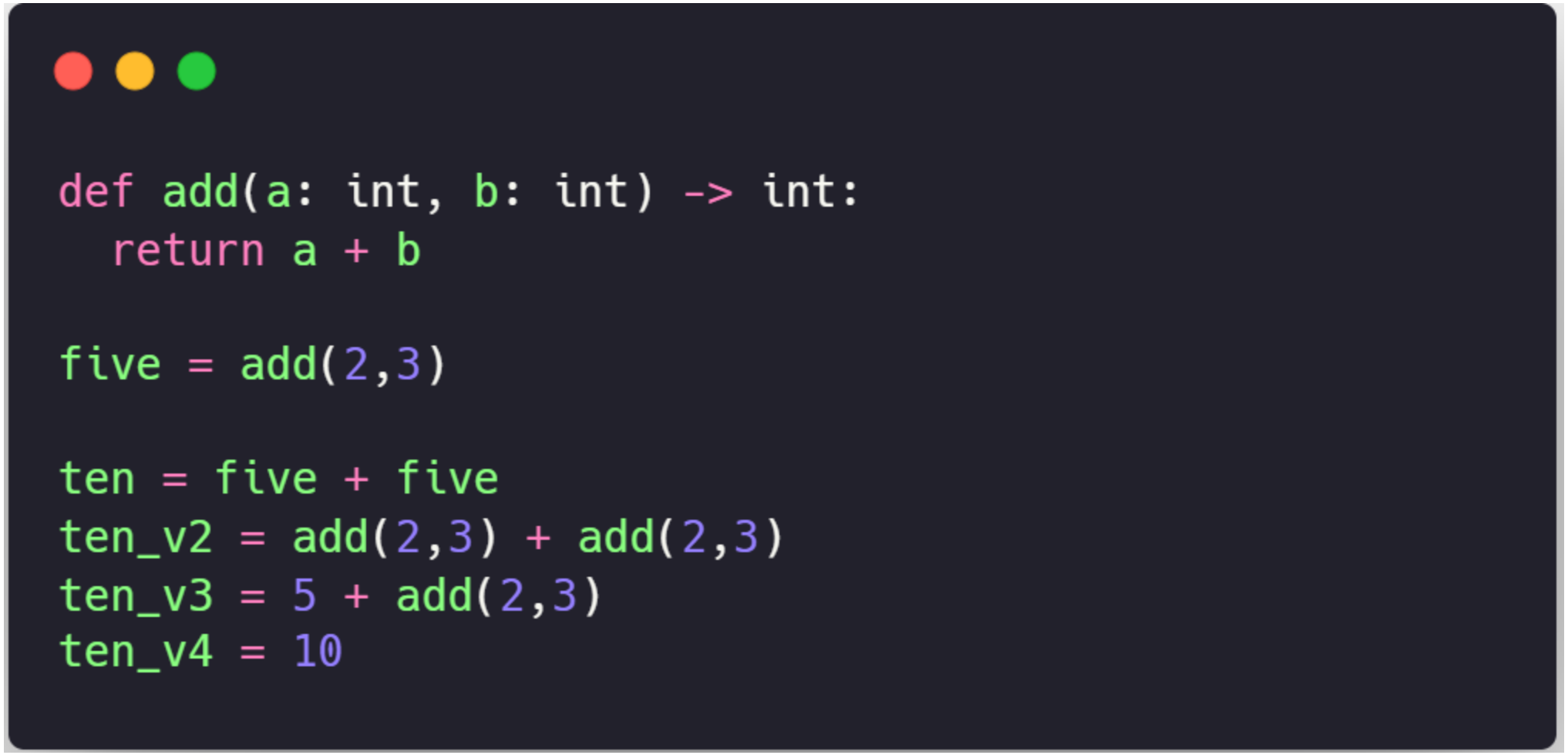
“Recursion”

In the functional programming paradigm, there is no for and while loops. Instead, functional programming languages rely on recursion for iteration. Recursion is implemented using recursive functions, which repetitively call themselves until the base case is reached.

```python
def factorial(n: int) -> int:
    return 1 if n <= 1 else n * factorial(n - 1)
```

“Referential Transparency”

The key differentiating feature of (pure) functional programs is that they provide referential transparency. An expression is said to be referentially transparent if it can be replaced with its corresponding value without changing the program’s behavior.



```python
def add(a: int, b: int) -> int:
  return a + b

five = add(2,3)

ten = five + five
ten_v2 = add(2,3) + add(2,3)
ten_v3 = 5 + add(2,3)
ten_v4 = 10
```

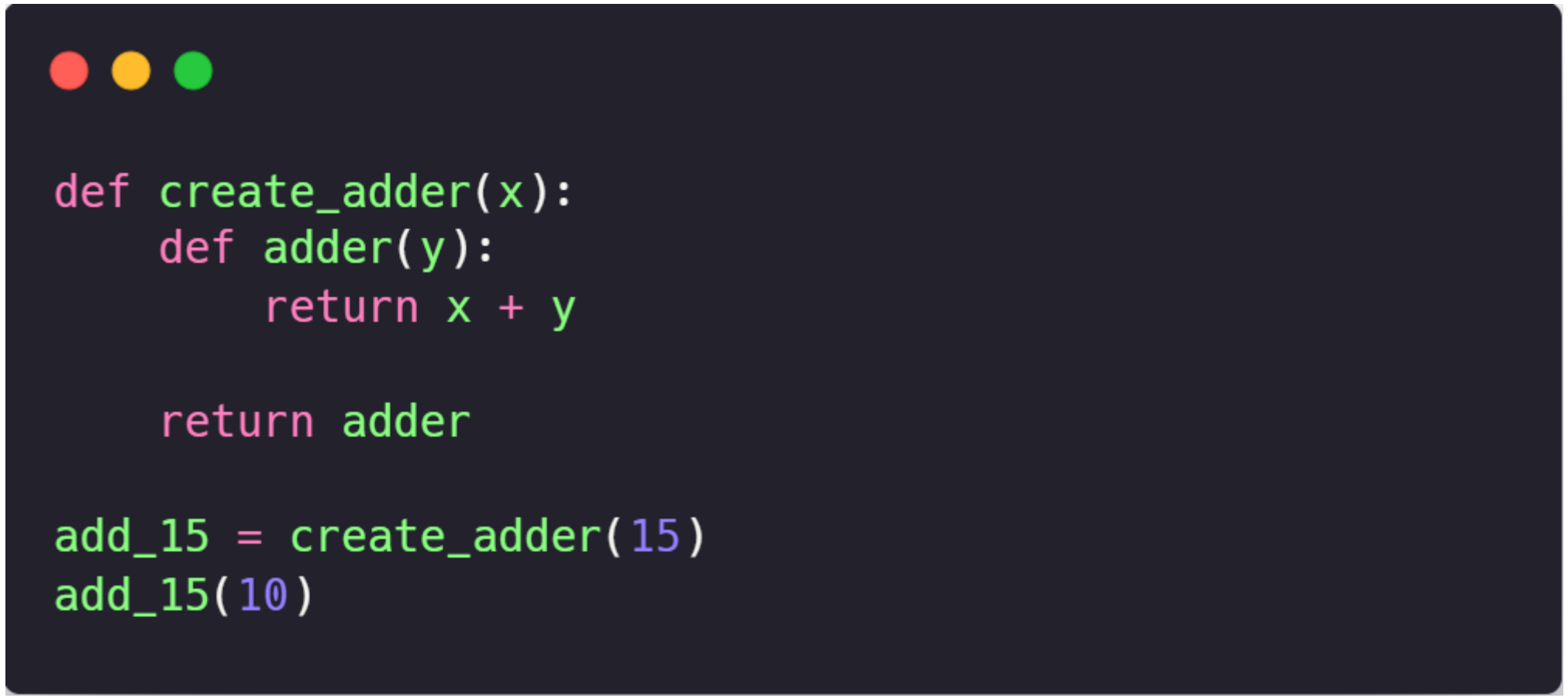
“Functions are First-Class”

Functions in the functional programming style are treated as variables. Hence, they are first-class functions. These first-class functions are allowed to be passed to other functions as parameters or returned from functions or stored in data structures.


def cube(x):
return x*x*x
my_cube = cube
my_cube(5)

“Higher-Order”

A higher-order function is a function that takes other functions as arguments and/or returns functions. First-Class functions can be higher-order functions in functional programming languages.



```python
def create_adder(x):
    def adder(y):
        return x + y

    return adder

add_15 = create_adder(15)
add_15(10)
```

“Variables are Immutable”

Variables are immutable i.e. it isn’t possible to modify a variable once it has been initialized. Though we can create a new variable, modifying existing variables is not allowed.

```
name = 'foo'
name = 'foo' + ' ' + 'bar'
```

```
name_1 = 'foo'
name_2 = 'bar'
name_3 = 'foo' + ' ' + 'bar'
```

“Eager vs Lazy Evaluation”

Eager evaluation: expressions are calculated at the moment that variables is assigned, function called.
Lazy evaluation: delays the evaluation of the expression until it is needed.

- Memory efficient: no memory used to store complete structures.
- CPU efficient: no need to calculate the complete result before returning.