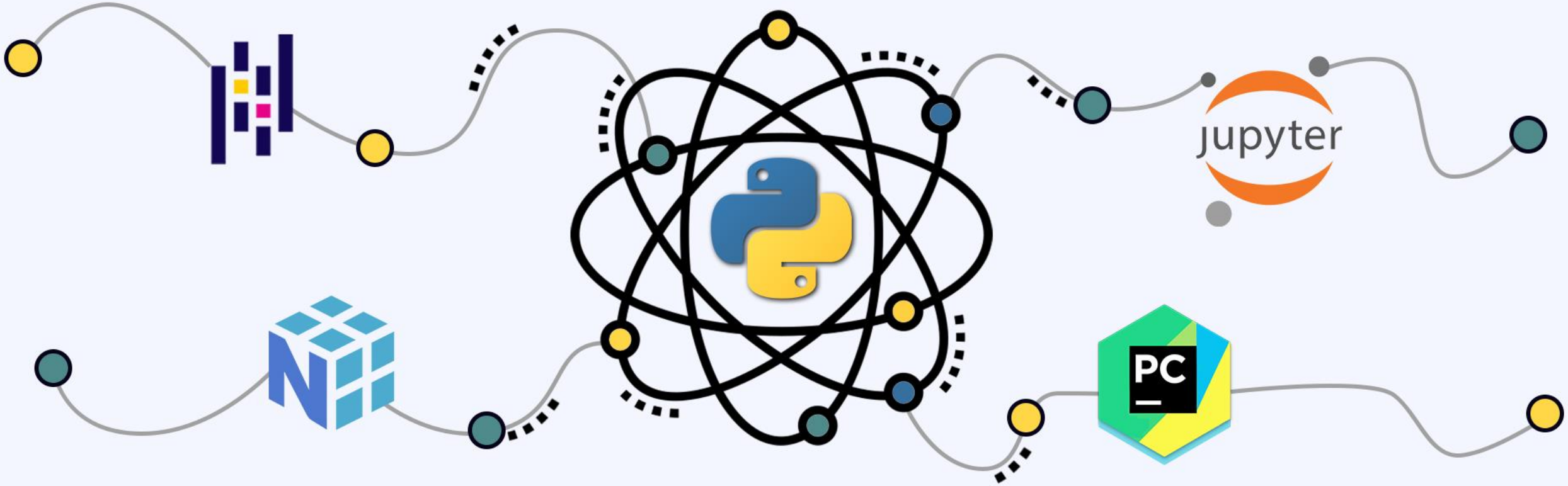
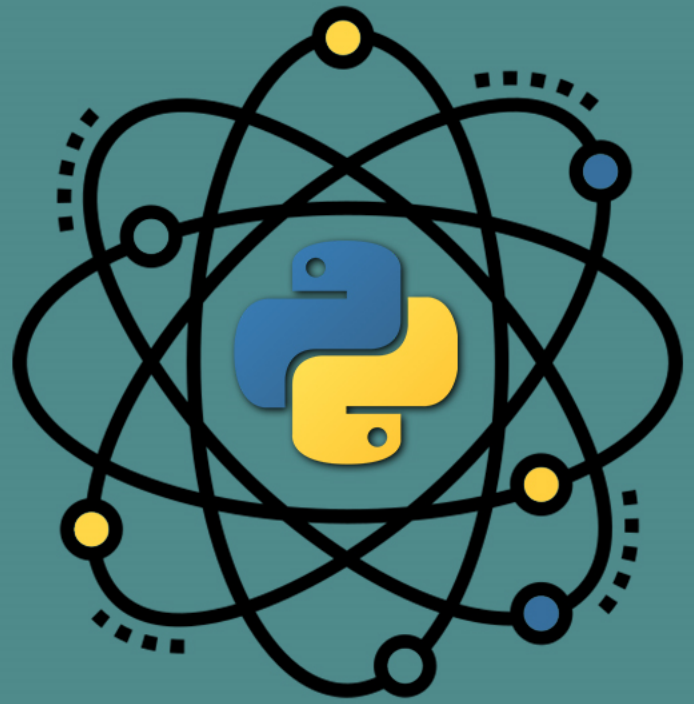


MOHAMMAD DEHGHANI PTYHON FOR DATA SCIENCE

زبان پایتون در علم داده

مدرس: محمد دهقانی



زبان پایتون در علم داده

مدرس:
محمد دهقانی

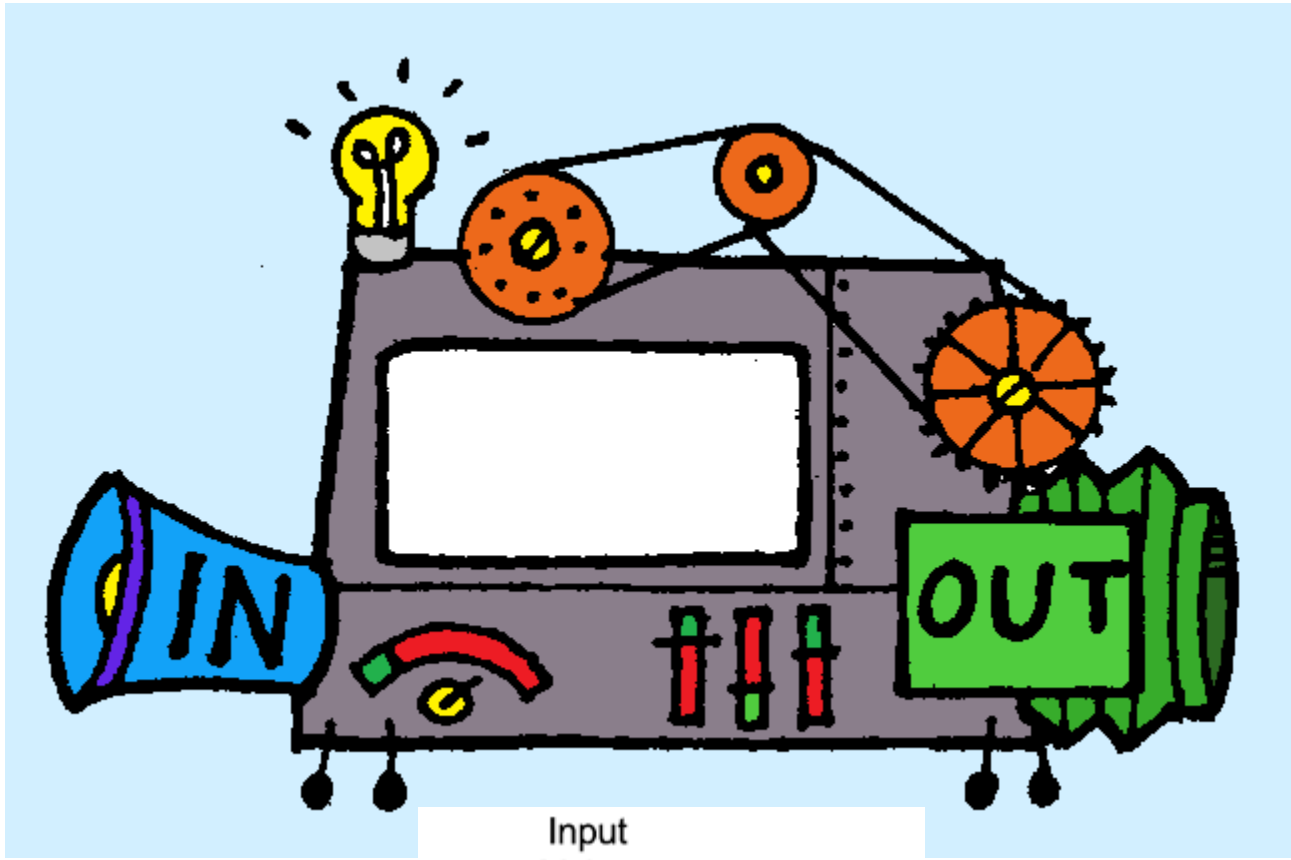
جلسه: دهم
موضوع: توابع

مواردی که در این جلسه بررسی می شوند:

1. تعریف تابع در Python
2. دریافت ورودی از کاربر
3. مفهوم Return در توابع
4. استفاده از چندین Return در توابع
5. Return چند مقدار در توابع
6. استفاده از توابع در توابع دیگر
7. دریافت تعداد نامحدودی ورودی در توابع
8. argها و kwarg ها
9. Docstring ها در پایتون
10. Scope در پایتون
11. Lambda Functions
12. توابع بازگشتی و مثال های آن
13. map
14. filter
15. reduce

Methods and Functions

- **Function —** a set of instructions that perform a task.
- **Method —** a set of instructions that are associated with an object.



Input Value

$$f(x) = x - 5$$

Function Name

Output Value

Difference between a function and a method

```
list1 = [3, 7, 2, 5, 4]

list1.sort()

print(list1)
```

sort( ) is a method

print( ) is a function

```
[2, 3, 4, 5, 7]
```

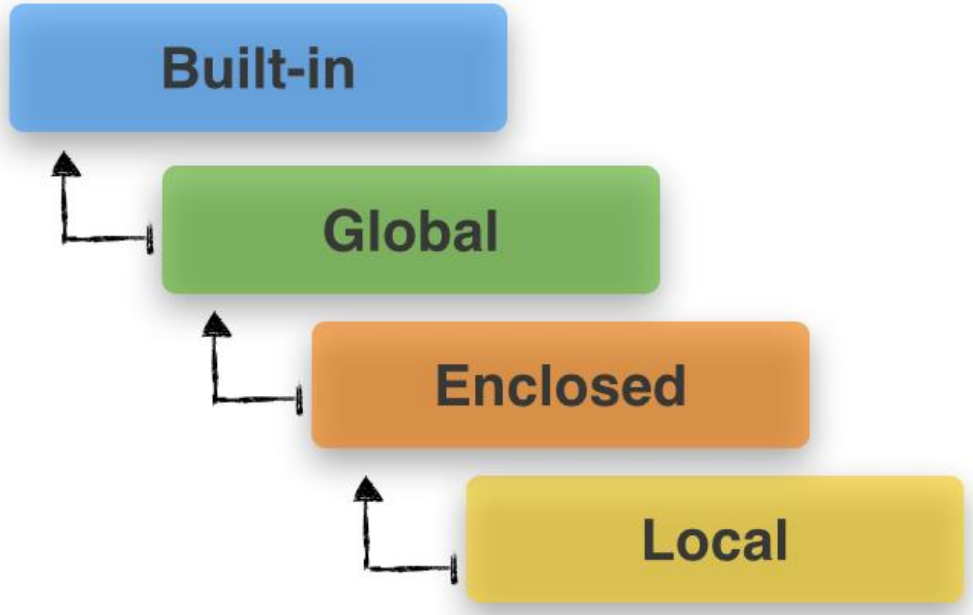
Functions

Python function in any programming language is a sequence of statements in a certain order, given a name. When called, those statements are executed. So we don’t have to write the code again and again for each [type of] data that we want to apply it to. This is called code re-usability.

Advantages of User-defined Functions

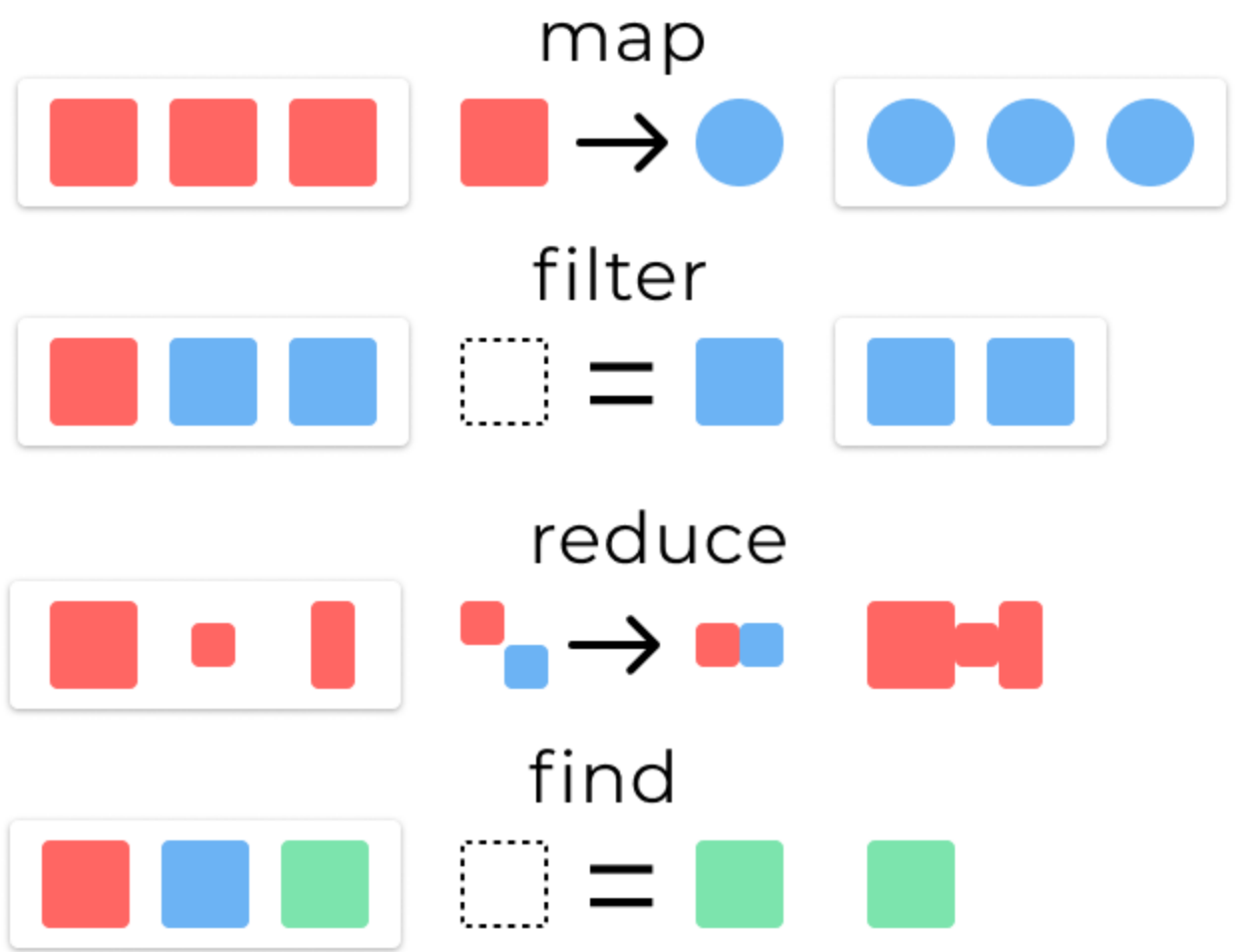
- This Python Function help divide a program into modules. This makes the code easier to manage, debug, and scale.
- It implements code reuse. Every time you need to execute a sequence of statements, all you need to do is to call the function.
- This Python Function allow us to change functionality easily, and different programmers can work on different functions.

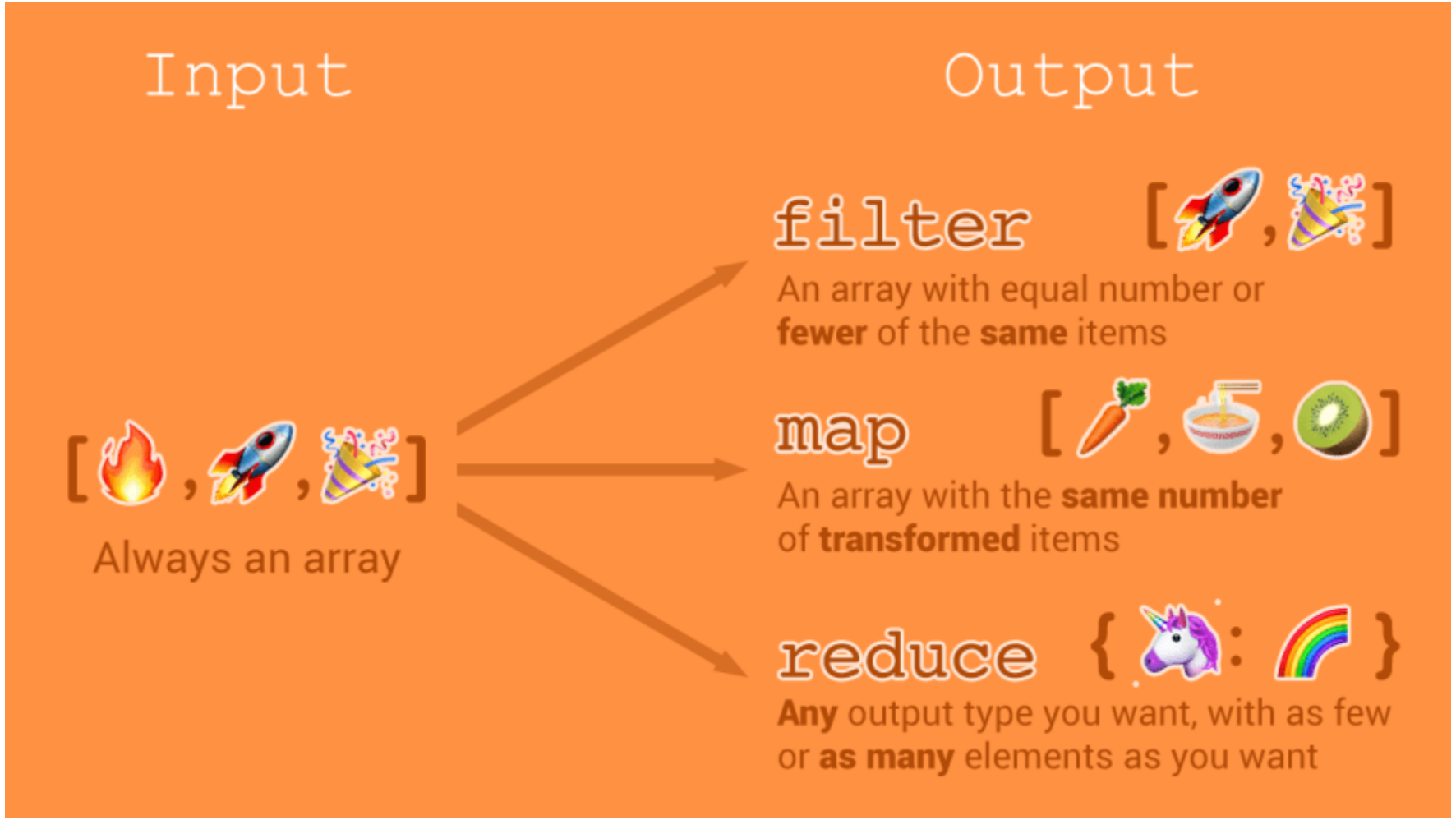
Variable Scope

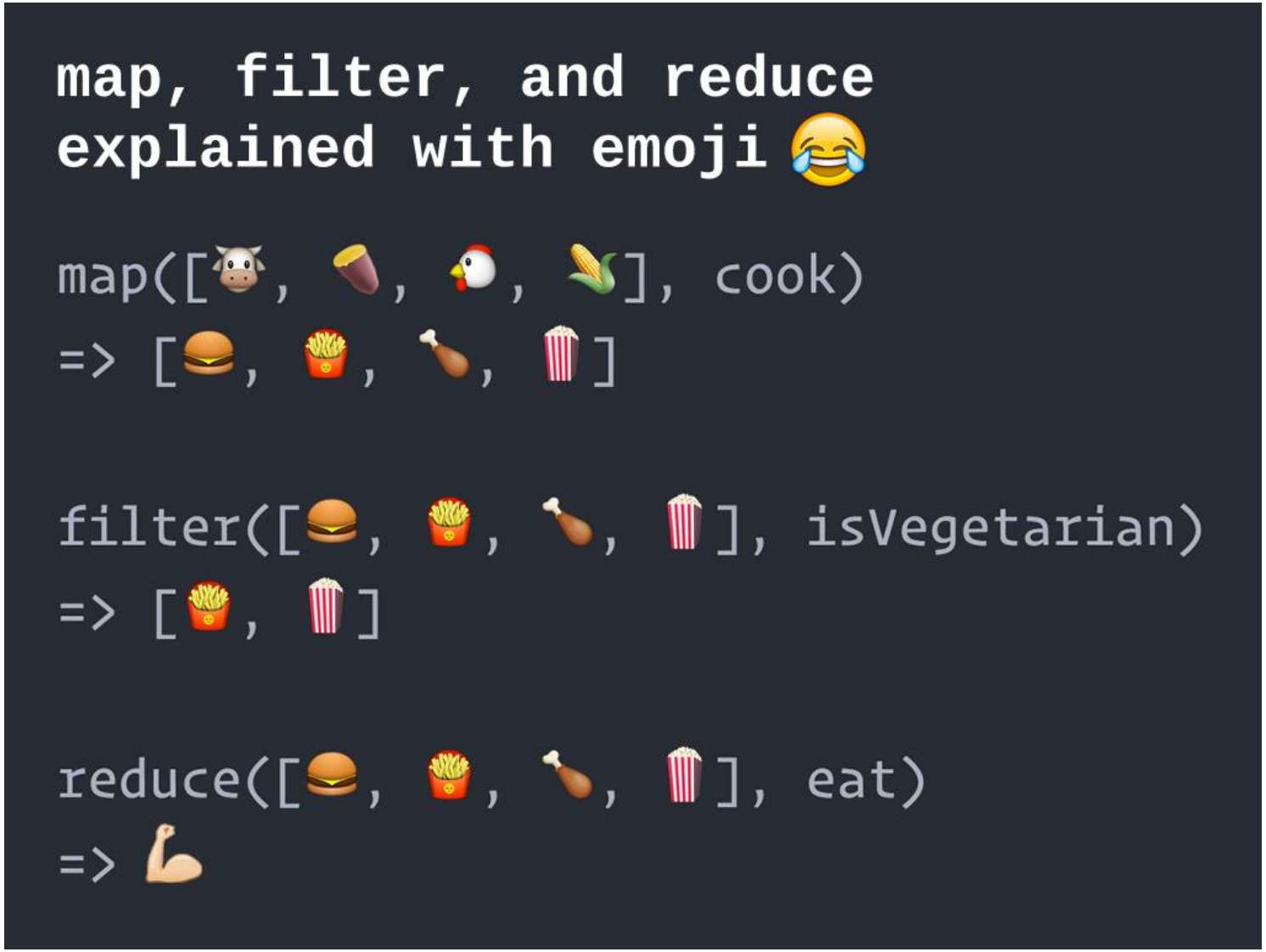


Advantages of User-defined Functions

The map(), filter() and reduce() functions bring a bit of functional programming to Python. All three of these are convenience functions that can be replaced with List Comprehensions or loops, but provide a more elegant and short-hand approach to some problems.


map
filter
reduce
find


Input
Output
[🔥, 🚀, 🎉]
Always an array
filter
[🚀, 🎉]
An array with equal number or
fewer of the same items
map
[🥕, 🍜, 🥝]
An array with the same number
of transformed items
reduce
{🦄: 🌈}
Any output type you want, with as few
or as many elements as you want


map, filter, and reduce
explained with emoji 😂
map([🐮, 🍠, 🐔, 🌽], cook)
=> [🍔, 🍟, 🍗, 🍿]
filter([🍔, 🍟, 🍗, 🍿], isVegetarian)
=> [🍟, 🍿]
reduce([🍔, 🍟, 🍗, 🍿], eat)
=> 💪

Recursive Functions

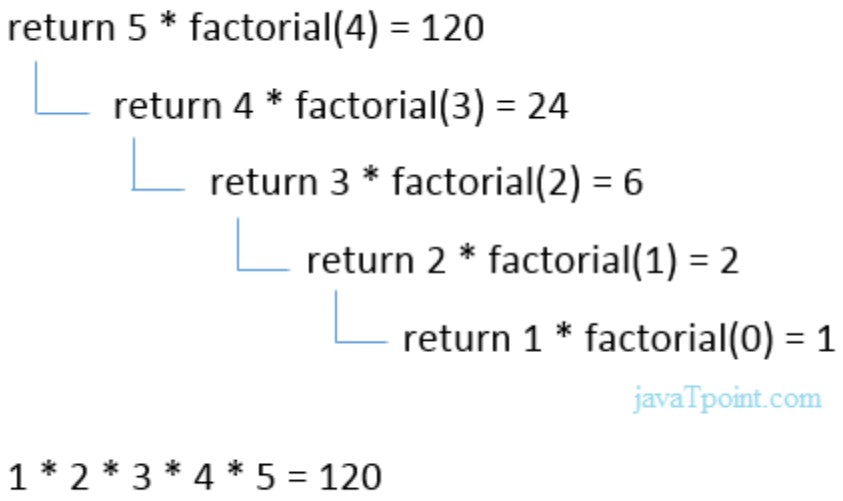
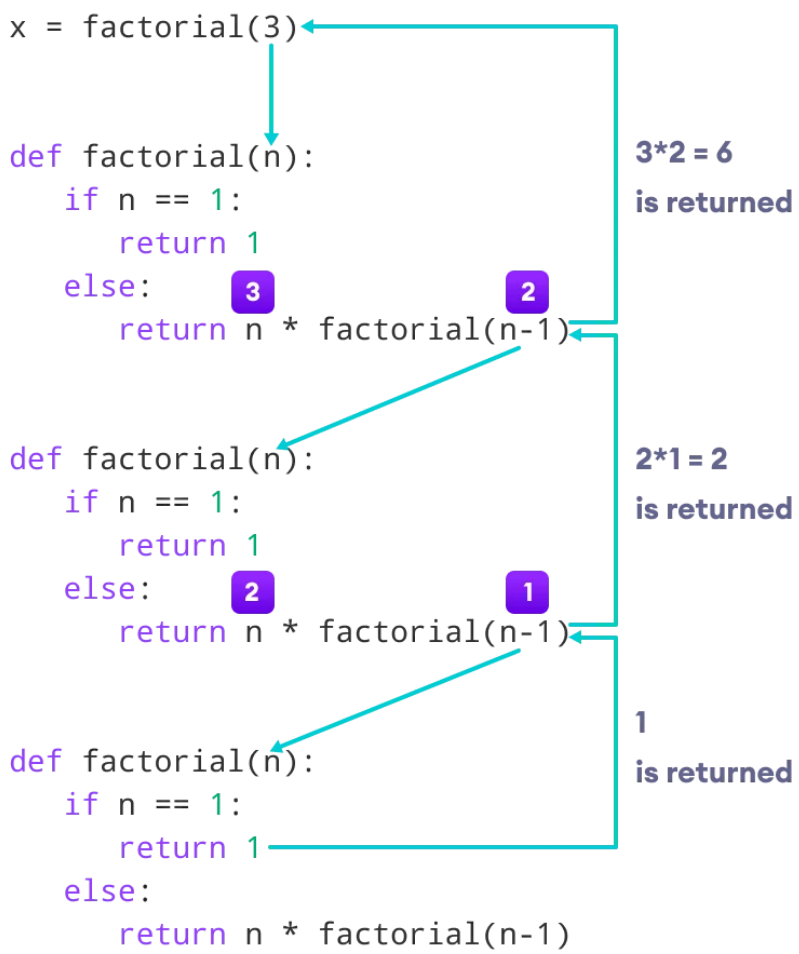
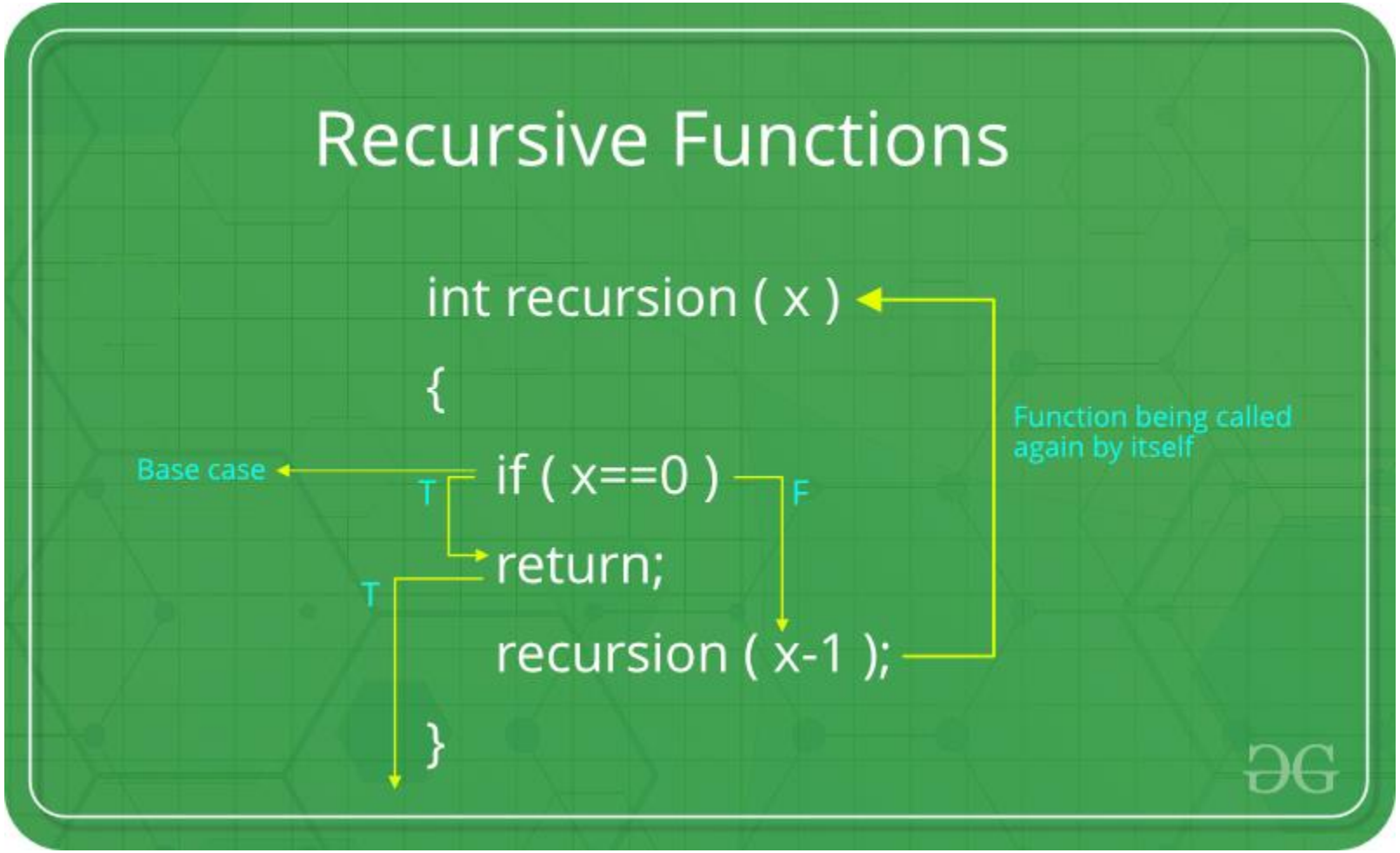


Fig: Recursion


x = factorial(3)
def factorial(n):
if n == 1:
return 1
else:
return n * factorial(n-1)
3
2
def factorial(n):
if n == 1:
return 1
else:
return n * factorial(n-1)
2
1
def factorial(n):
if n == 1:
return 1
else:
return n * factorial(n-1)
3*2 = 6
is returned
2*1 = 2
is returned
1
is returned


Recursive Functions
int recursion (x)
{
if (x==0)
return;
recursion (x-1);
}
Base case
T
F
T
Function being called
again by itself

Advantages and Disadvantages

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.