Newton School of Technology

20

Sets in Python

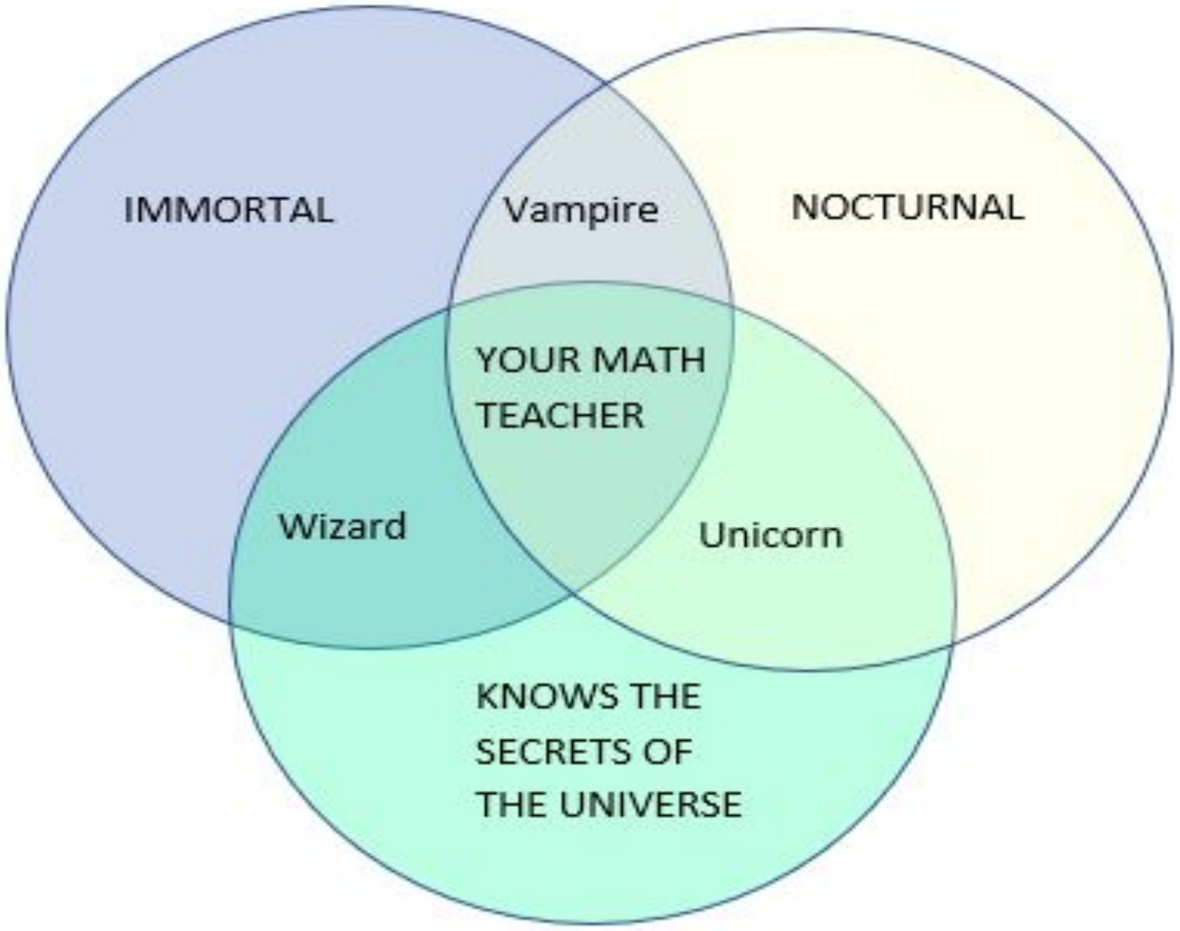
by Gladden Rumao

C02: Problem Solving with Programming

Newton School of Technology

Mathematical Set Operations

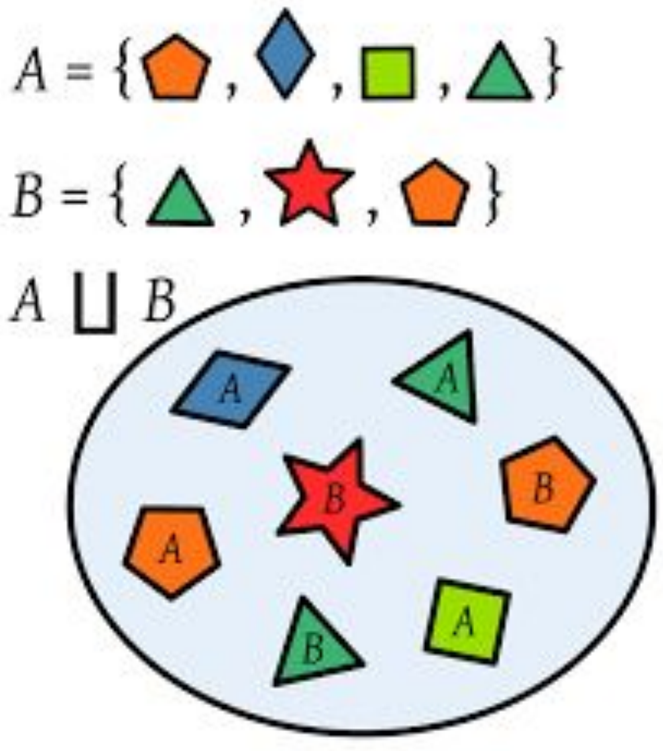
Exploring Types of Set Operations:



Union

Newton School of Technology

Combines **all unique elements from two sets** & returns a **new set** containing all elements that are **in either of the sets.**

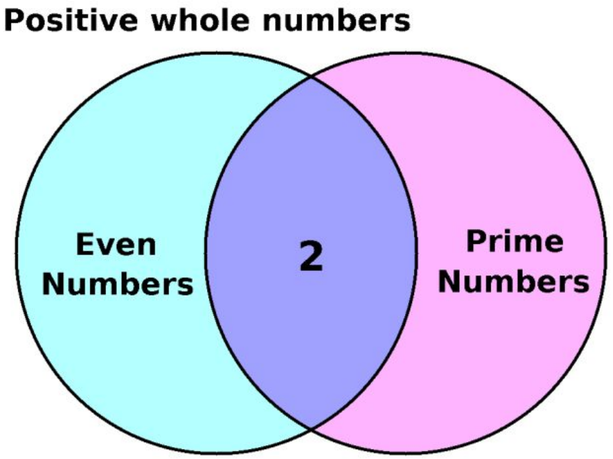



```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

union_set = set1 | set2
print( union_set)

union_set = set1.union(set2)
print( union_set)
```

Intersection

Finds **common elements between two sets** & returns **a new set** containing elements that are present in both sets.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

intersection_set = set1 & set2
print( intersection_set)

intersection_set = set1.intersection(set2)
print(intersection_set)
```



Output:

{3}

Differences

Newton School of Technology

Finds elements that are **in the first set but not in the second set**. The result is a **new set** containing elements that are only in the first set.

```
# Define two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# Difference of sets
difference_set = set1 - set2
print("Difference:", difference_set)

# Alternative method
difference_set = set1.difference(set2)
print("Difference (using difference()):", difference_set)
```

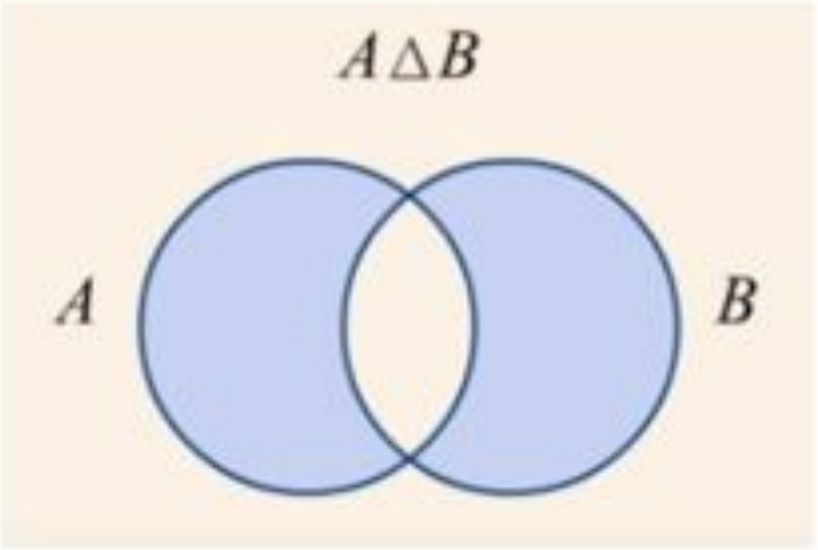
Symmetric Differences

Newton School of Technology

Returns a set containing elements that **are in either of the sets but not in both** or **union of the differences** of the two sets.

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

sym_diff1 = set1 ^ set2
sym_diff2 = set1.symmetric_difference(set2)

print(sym_diff1)
print(sym_diff2)
```

Output: {1, 2, 5, 6}



What Will You Do?:

Imagine you have a **set of unique numbers**, and you want to create a copy of this set to **modify it without altering the original set.**

Copying a set:

The **copy()** method creates a **shallow copy** of the set. It creates a **new set** with the **same elements** as the original set, but it is a **different object in memory.**

```
original_set = {1, 2, 3, 4, 5}
copied_set = original_set.copy()

print("Original Set:", original_set)
print("Copied Set:", copied_set)

copied_set.add(6)

print("Original after modification:", original_set)
print("Copied after adding 6:", copied_set)
```

Output

```
Original Set: {1, 2, 3, 4, 5}
Copied Set: {1, 2, 3, 4, 5}
Original after modification: {1, 2, 3, 4, 5}
Copied after adding 6: {1, 2, 3, 4, 5, 6}
```

Newton School of Technology

Set Comprehension

Set Comprehension

Newton School of Technology

Set comprehension is a **concise way to create sets** in Python **using an expression** on **elements** from an **iterable**, optionally **filtered by a condition.**

Syntax:

```
{expression for item in iterable if condition}
```

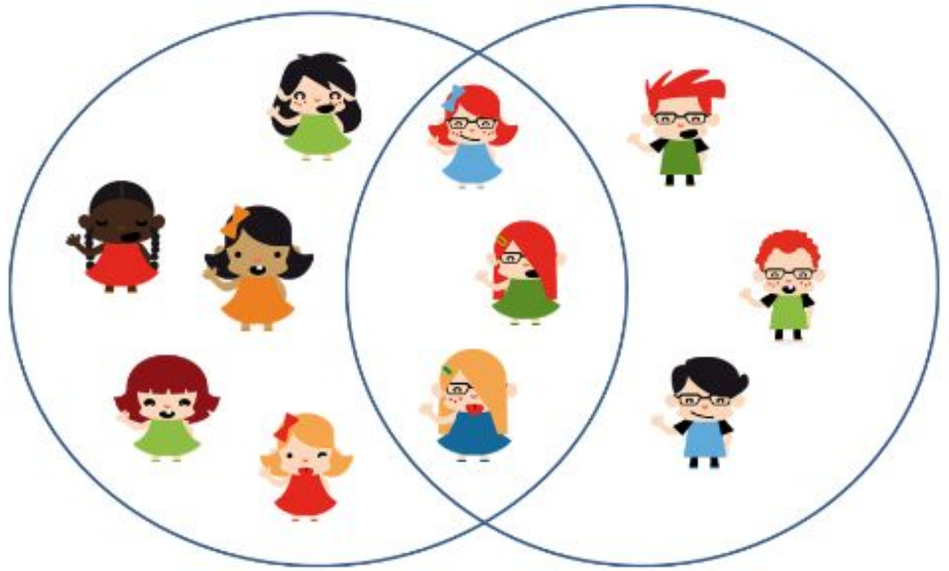
Example:

```
squared_set = {x*x for x in range(1, 6)}
```

Output: {1, 4, 9, 16, 25}

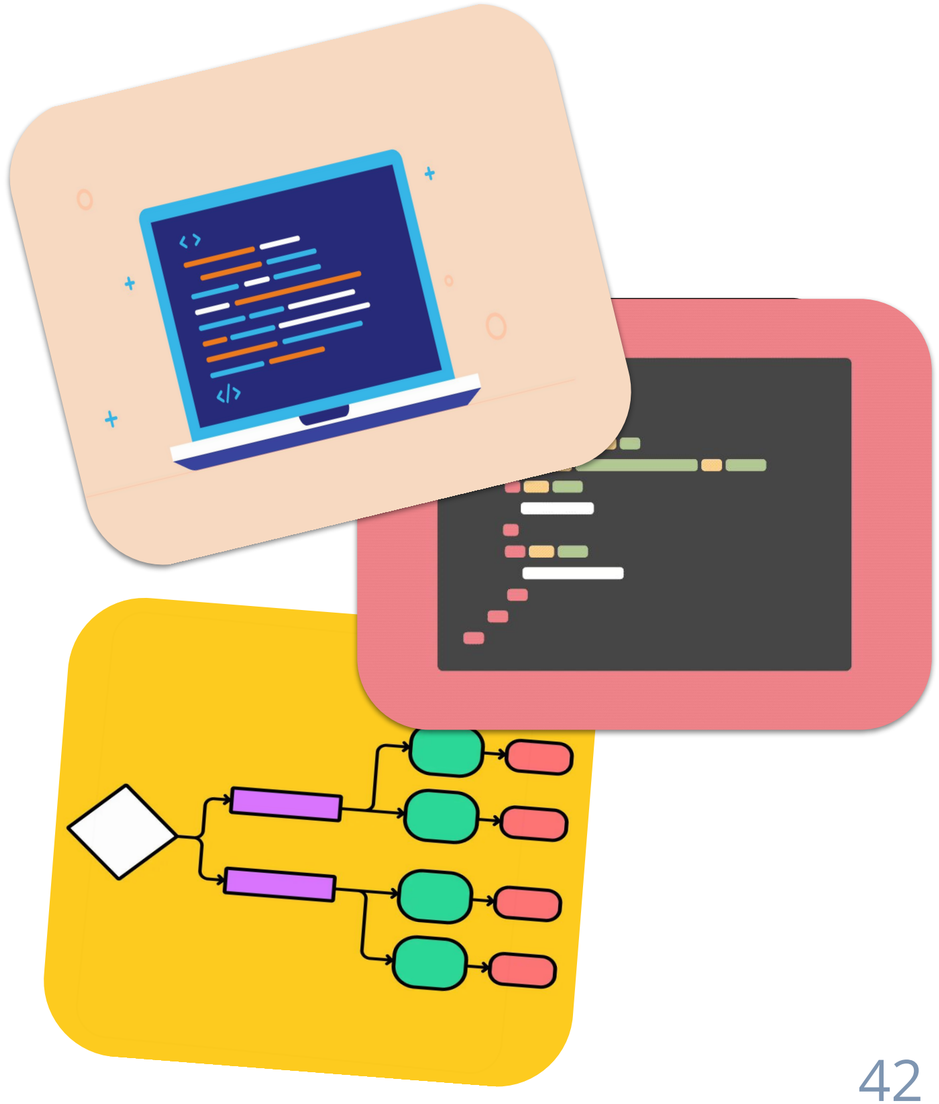
Real-life Use Cases

- Tracking **Unique Visitors** on a Website
- Finding **Common Interests** in Social Networks
- **Managing emails** of Subscribers/ Email Campaign
- **Inventory Management**



Summary

- **Sets:** Unordered, unique elements
- **Creation:** {}, set()
- **Membership Testing**:` in`
- **Access:** Iterate
- **Basic Ops:** add(), remove(), discard(), pop(), clear()
- **Set Ops:** Union |, Intersection &, Difference -, Symmetric Diff ^
- **Methods:** copy(), in
- **Comprehensions:** {x*x for x in range(1, 6)}



Newton School of Technology

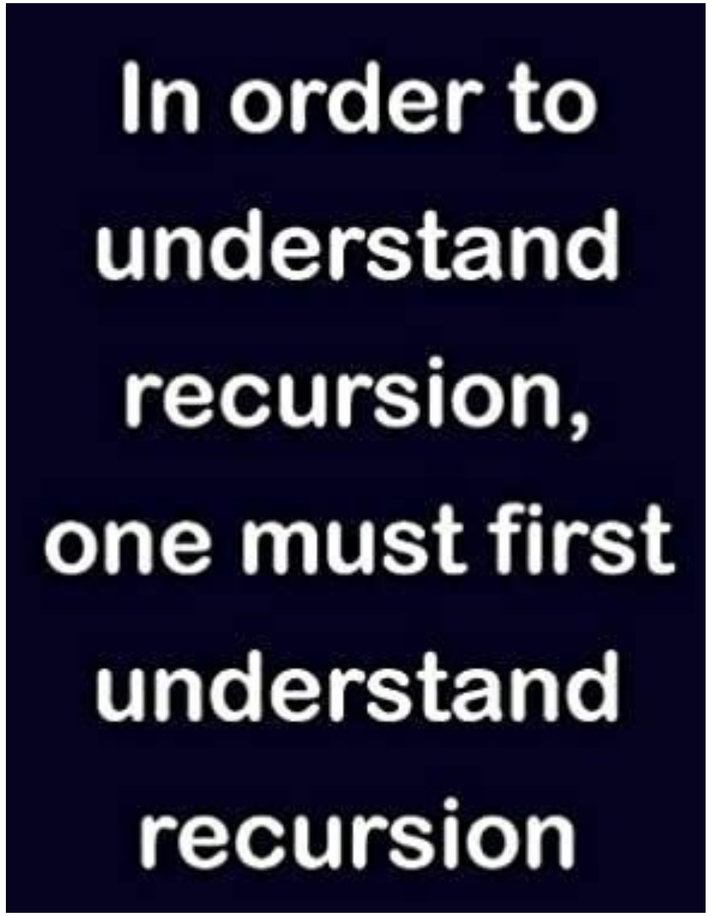
Recursion Part-1

by Gladden Rumao

CSA101 : Problem Solving with Programming

Quick Activity:

Students to search recursion in google



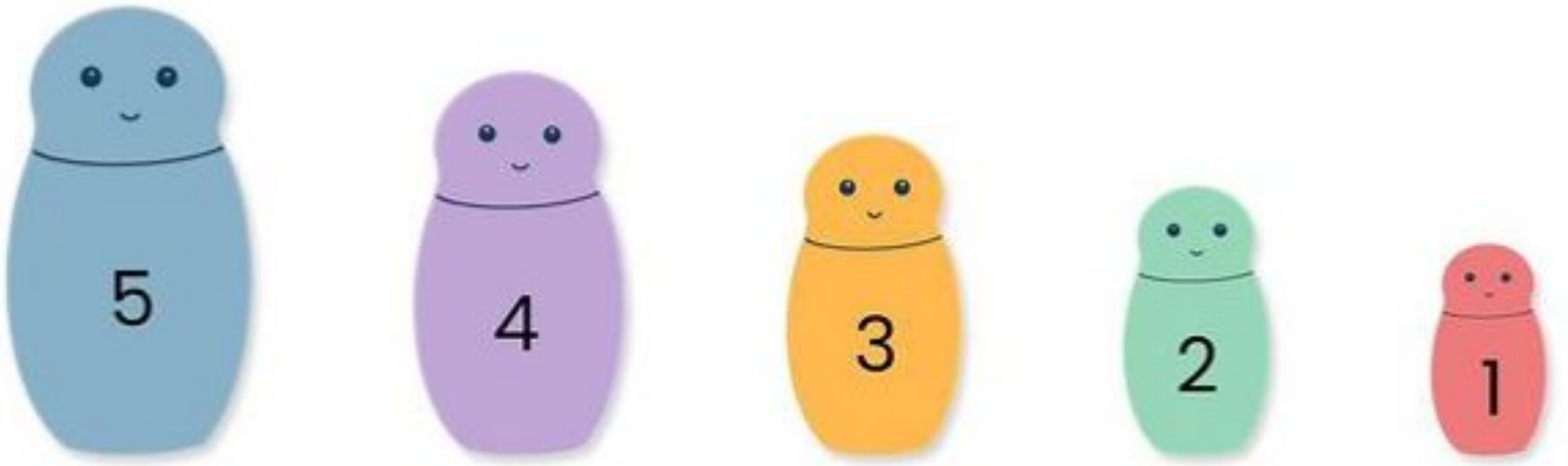
Definition

Recursion is a programming technique where a **function calls itself** to solve a problem.

Instead of solving the entire problem at once, the function divides it into smaller subproblems.

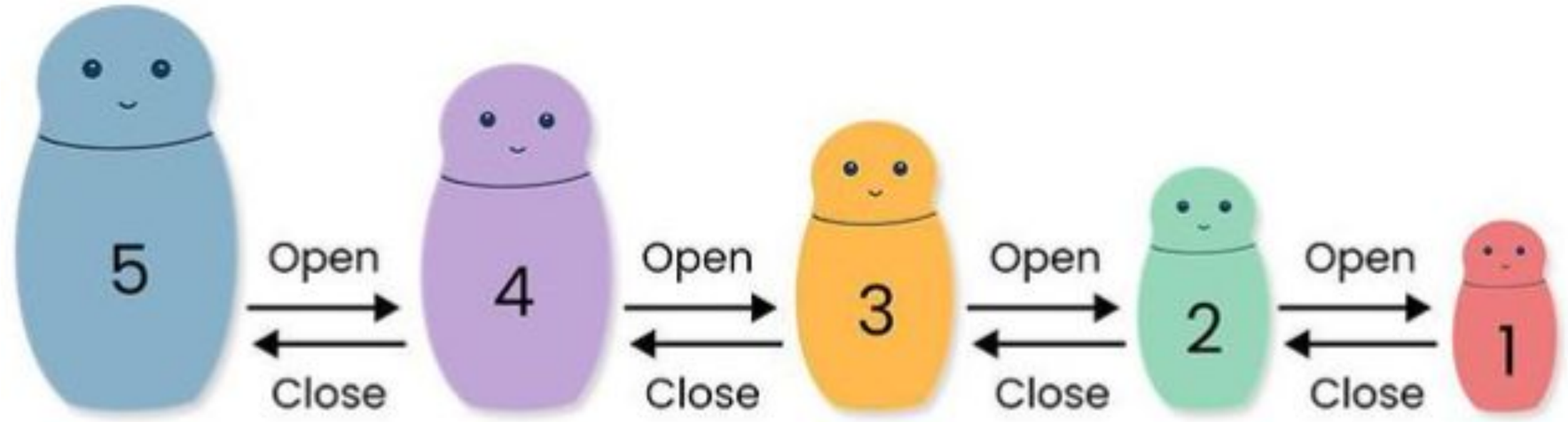
Recursion through Example

Russian nesting dolls is a sequence of similar dolls inside each other that can be opened.
Each time you open a doll a smaller version of the doll will be inside , until you reach the innermost doll which cannot be opened



Russian Nesting Dolls

Recursion is like opening the dolls one by one and putting them back together

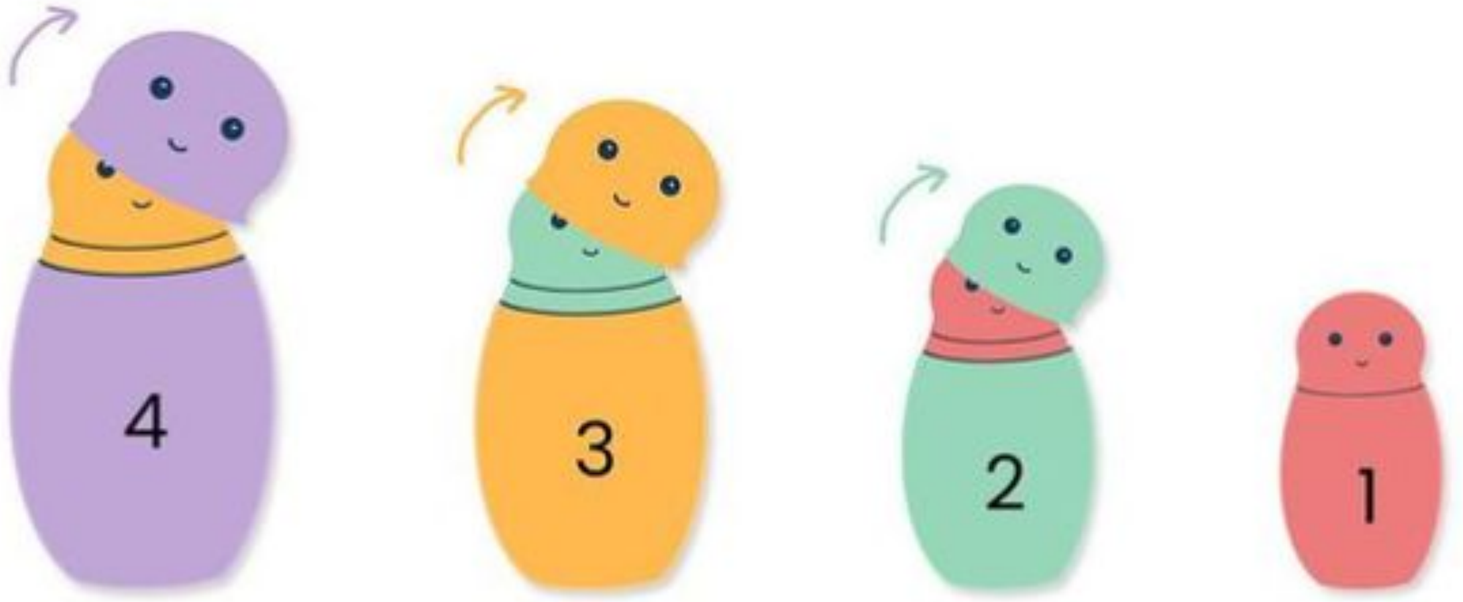


Russian Nesting Dolls

The function divides it into smaller subproblems

Recursion through Example

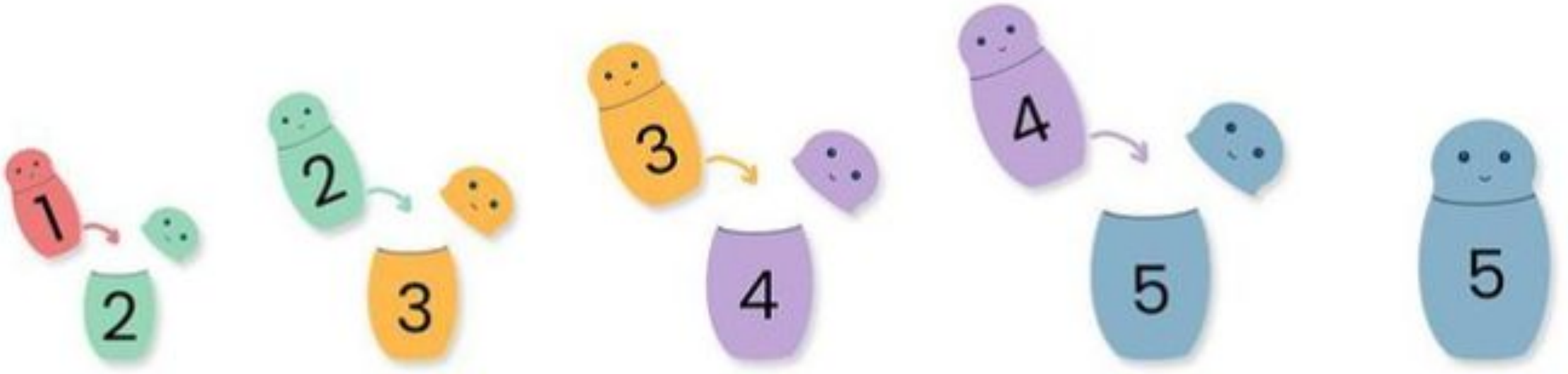
Repeating the process of opening and finding smaller dolls until the base condition



Base Condition

Recursion through Example

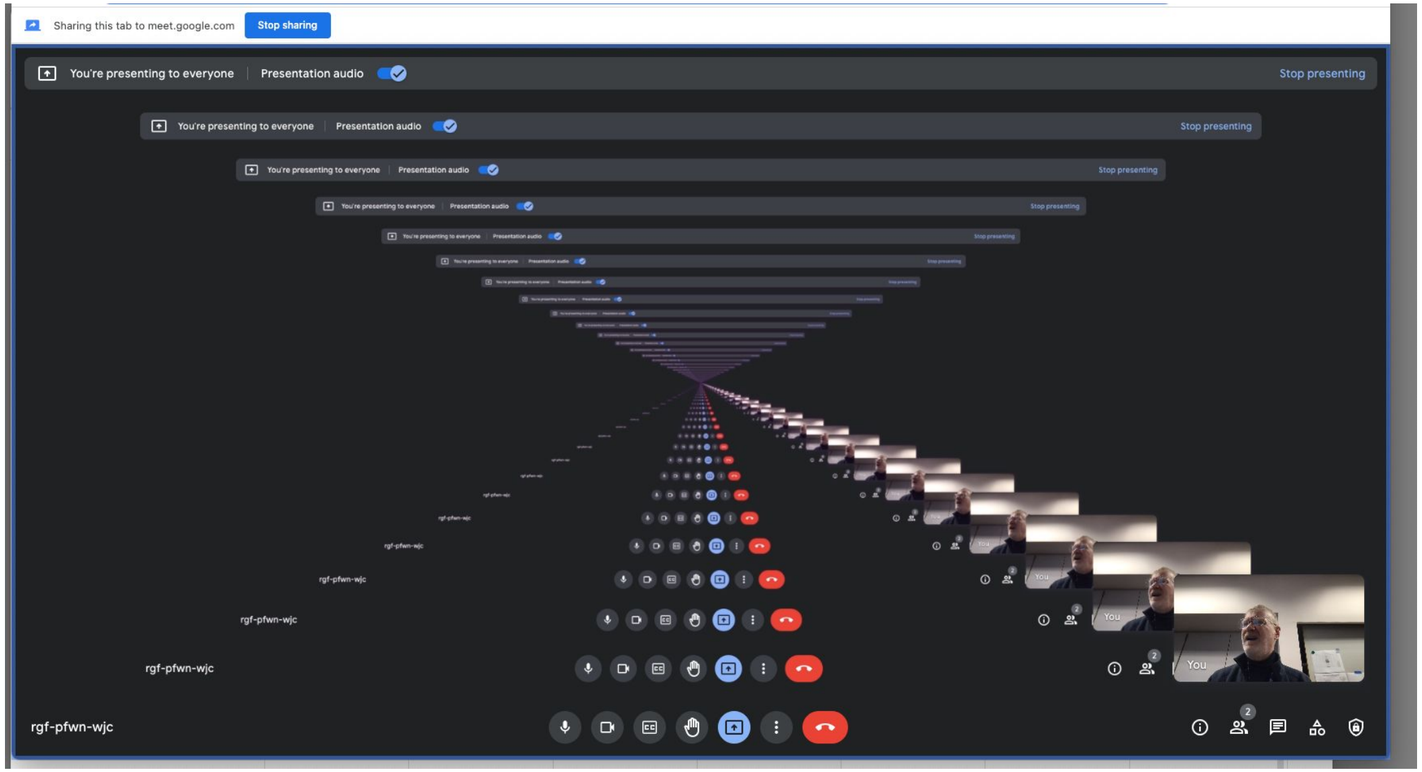
Putting the dolls back together , returning the smaller ones to bigger one



Recursion

Infinite Recursion

```
1  def recurse(i):
2      print(i)
3      recurse(i + 1)
4
```

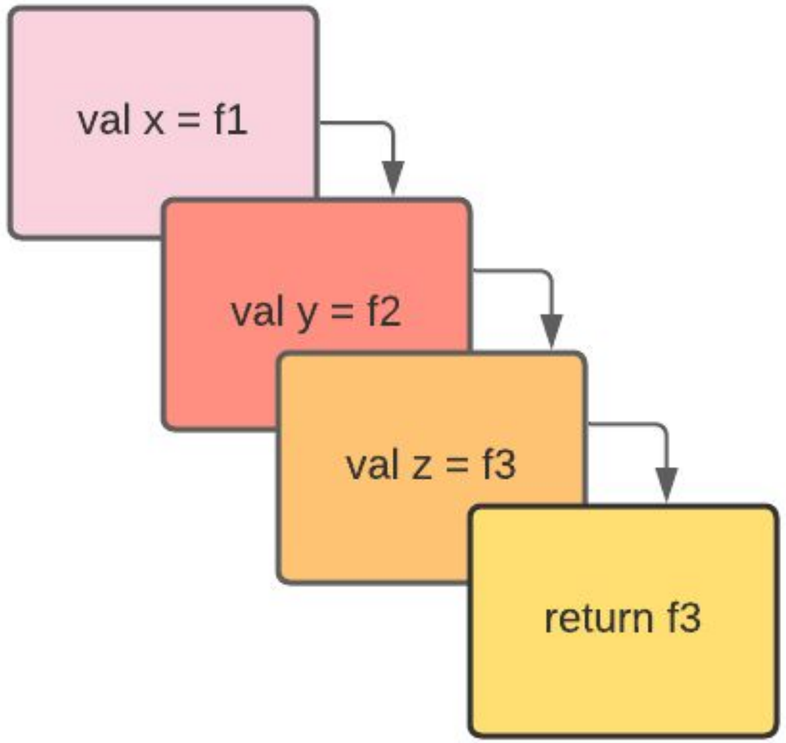


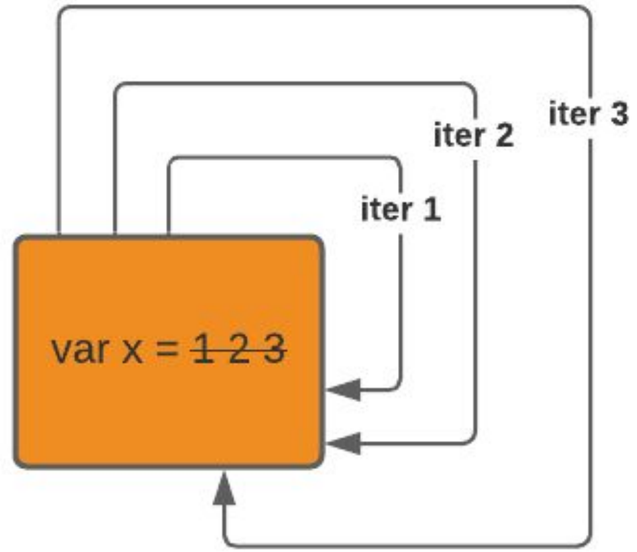
Base Case

Prevent this, we need to add in a stopping condition, or ***base case***

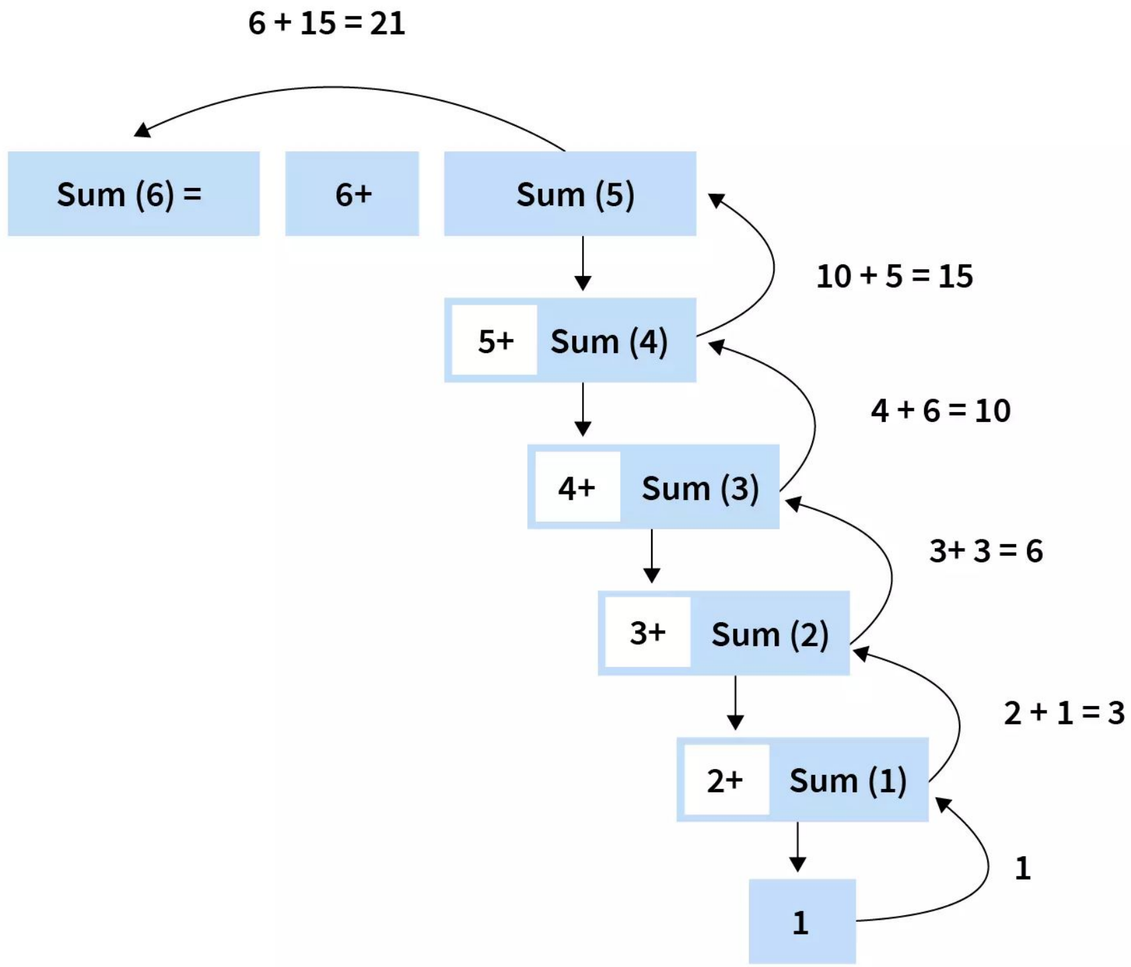
```python
def recurse(i, limit=10):
    if i > limit:  # Base case to stop recursion
        return
    print(i)
    recurse(i + 1) # recursive case

recurse(1)
```

Recursion Vs Iteration





Example- Sum of numbers- 1 to n



Example- Sum of numbers- 1 to n

```python
def add_numbers(n):
    if n > 0:  # Use n > 0 instead of n != 0
        return n + add_numbers(n - 1)  # Recursive call
    else:
        return 0  # Base case to stop recursion

# Input from user
num = int(input("Enter a positive integer: "))
print("Sum =", add_numbers(num))
```

Output

```
Enter a positive integer: 5
Sum = 15
```

Example- Sum of numbers- 1 to n

```python
def add_numbers_iterative(n):
    sum = 0  # Initialize sum to 0
    for i in range(1, n + 1):  # Loop from 1 to n
        sum += i  # Add each number to the sum
    return sum

# Input from user
num = int(input("Enter a positive integer: "))
print("Sum (Iterative) =", add_numbers_iterative(num))

```

Output

```
Enter a positive integer: 5
Sum (Iterative) = 15
```

Recursion Mechanism

Online Python Tutor - Visualize program execution

When to use Recursion

- When a problem can be broken down into smaller, similar subproblems.
- When calculating values that can be defined in terms of themselves
  Example- factorial of a number (n!)
- When dealing with data that has multiple layers of nesting, such as lists within lists or objects within objects.
- When generating combinations, permutations, or subsets of a set.

Advantages and Disadvantages

- **Advantages:**
  - Simplified Code
  - Natural Fit for Certain Problems
  - Ease of Implementation

- **Disadvantages:**
  - Performance Issues
  - Higher Memory Usage
  - Base Case Complexity

Newton School of Technology

Question – Print 1 to N recursively

Newton School of Technology

Question – Print N to 1 recursively

Newton School of Technology

Sum of numbers – 1 to N

Thank You!