

Recursion

Textbook

Recursion



This lesson covers recursion, and how you can use it to simplify your programs.

Overview

Recursion is a very abstract concept that usually takes beginners some time to understand. If it takes you a while to wrap your mind around it, don't worry! It can take some work to figure out.

In short, recursion is writing functions that call themselves. You might wonder why this is necessary or useful, and you'll learn why in this lesson.

Recursive Santa Claus

To help illustrate the concept of recursion, let's use Santa and his elves as an example. .

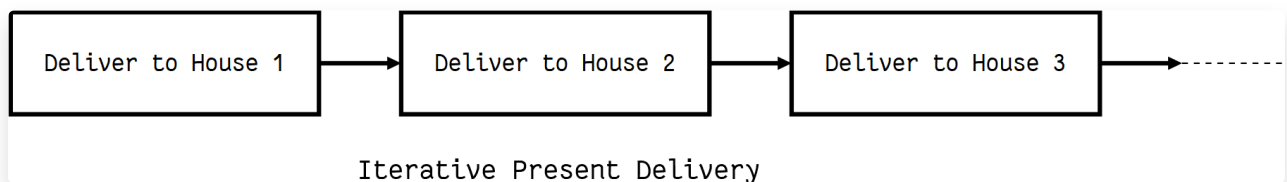
Santa's Iterative Plan

Have you ever wondered how Christmas presents are delivered? Most people usually imagine Santa going to each house, one at a time. If we were to write a Python program to describe this process, it would probably look like this:

```
1 houses = ["Eric's house", "Kenny's house", "Kyle's house", "Stan's house"]
2
3 def deliver_presents_iteratively(houses):
4     for house in houses:
5         print("Delivering presents to", house)
6
7 deliver_presents_iteratively(houses)
```

Try it!

As you can see, the iterative Santa goes to each house one by one. This is represented in the image below:



This would take a long time! Some estimates put the total number of houses in the world at over 1.5 billion. Assuming 10 seconds per house, it would take Santa hundreds of years to visit each one. There's no way he does that in one night!

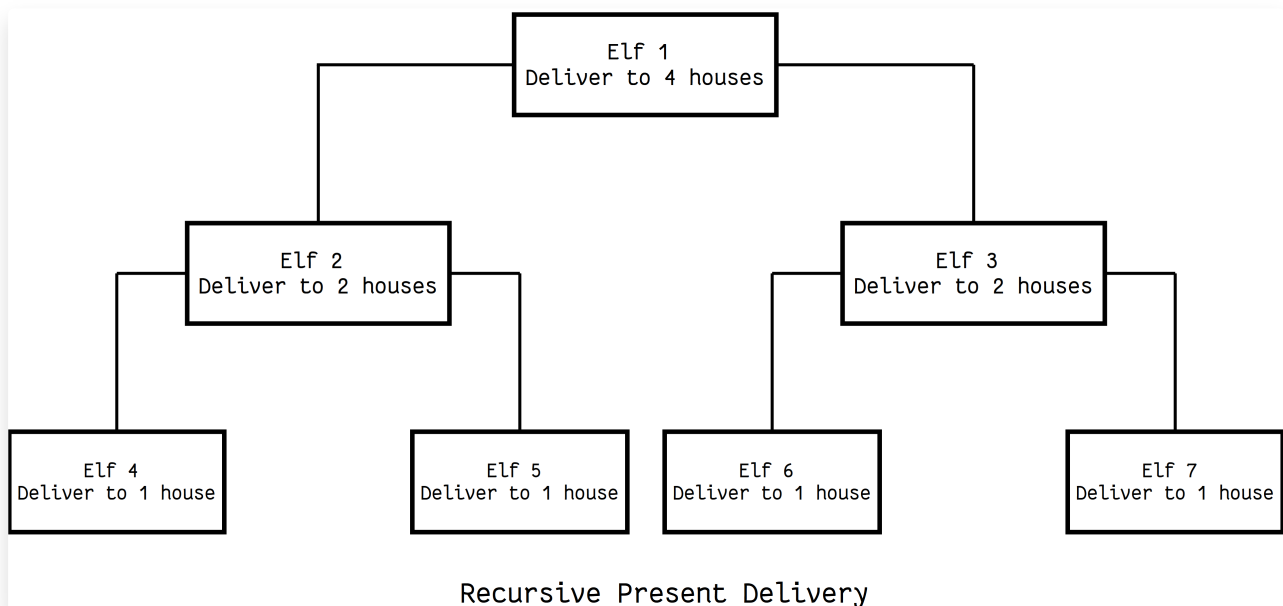
Learn more about this analogy at www.realpython.com.

Santa's Recursive Plan

Luckily for Santa, there's a [recursive](#) way he can do this where he uses his elves as assistants. Here's the basic algorithm for this method:

1. Appoint an elf and give all the work to him.
2. Assign titles and responsibilities to the elves based on the number of houses for which they are responsible.
3. If an elf is responsible for more than one house, he is a manager and can divide his work among two other elves.
4. If an elf is only responsible for one house, he is a worker who delivers presents to that house.

The diagram below illustrates this algorithm:



Here's what this recursive function would look like in Python. The purpose of this function is to break up the list into smaller chunks to work with.

```
1 houses = ["Ahmed's house", "Clark's house", "Xin Li's house", "Melissa's house"]
2
3 # Each function call represents an elf doing his work
4 def deliver_presents_recursively(houses):
5     # Worker elf doing his work
6     if len(houses) == 1:
7         house = houses[0]
8         print("Delivering presents to " + house)
9
10    # Manager elf doing his work
11    else:
12        mid = len(houses) // 2
13        first_half = houses[:mid]
14        second_half = houses[mid:]
15
16        # Divides his work among two elves
17        deliver_presents_recursively(first_half)
18        deliver_presents_recursively(second_half)
19
20 deliver_presents_recursively(houses)
```

Try it!

Let's talk about what's happening here. We see a big function called `def deliver_presents_recursively()`.

Inside this function, we have an if statement that checks to see if the length of the list named `houses` is 1. If it is, we print `"Delivering presents to " + house`.

If the length of the list named `houses` is not equal to 1, we create a variable named `mid` and set it equal to the middle of the list. (found by taking the length and dividing by 2—the two `//` symbols means it will discard the remainder.)

Now that we know where the middle is, we use it to find the `first_half` and the `second_half` of the list.

Now we can call the function named `deliver_presents_recursively()` again but on specifically the `first_half` and the `second_half`.

Notice the **last two lines of the function**. The function named `deliver_presents_recursively()` is being called on itself, *inside* itself. This is [recursion](#) – when a function calls itself.

Then the function runs again but with just the `first_half`. Which then gets cut in half and continues all the way, cutting in half again and again until it gets to just one item in the list.

Why Recursion?

You might wonder why this is important to know if there are other ways to get to that result. With big, complicated structures, recursion can really simplify the code process. Instead of calling a function hundreds of times, you can call it once and tell it to call itself until a condition is met.

This saves programmers a lot of time and resources.

Checkpoint

Recursion

You are the owner of a zoo and need to make sure that all the animals have been fed. You have a team of animal specialists help you. You decide to create a Python function to help organize everybody and assure each animal gets the food they need.

Create a recursive function!

Example output:

The panda has been fed

The lion has been fed

The giraffe has been fed

The tiger has been fed

The elephant has been fed

The monkey has been fed

The fish has been fed

The snake has been fed

The bearded dragon has been fed

The koala has been fed

Example Output

```
animals = ['panda', 'lion', 'giraffe', 'tiger', 'elephant', 'monkey', 'fish', 'snake', 'bearded dragon', 'koala'] def feeding(animals): if len(animals) == 1: animal = animals[0] print("The " + animal + " has been fed") else: mid = len(animals) // 2 first_half = animals[:mid] second_half = animals[mid:] feeding(first_half) feeding(second_half) feeding(animals)
```

Requirements:

- Create a list named `this_list` with the name of at least 10 animals inside.
- Create a function named `feeding` and put the appropriate parameter inside.

- Inside the function named `feeding` , create an if statement that checks to see if the length of the list named `this_list` is equal to 1.
- If the length of `this_list` is equal to 1, print "The _____ has been fed" with the name of the animal in the blank spot. (Concatenate the code `this_list[0]` into the blank spot)
- If the length of `this_list` is not equal to 1, find the middle of the list and assign it to a variable.
- Inside the else statement, create a variable named `first_half` and assign it to the first half of the list named `this_list` .
- Inside the else statement, create a variable named `second_half` and assign it to the second half of the list named `this_list` ..
- Inside the else statement, call the function named `feeding` with just the first half of the list as a parameter.
- Inside the else statement, call the function named `feeding` with just the second half of the list as a parameter.
- At the very end, make sure to call the function named `feeding` with the list named `this_list` as an argument!

Questions (3)

1. Why do programmers use recursion when there are other ways to get the same result? Select all that apply.

SELECT MULTIPLE

Select all that apply:

- A. More calculations can be done with less code.
- B. It saves time and resources.
- C. It helps break down complicated tasks into easier chunks.
- D. It enlists the help of more programmers, making their jobs easier.

2. What is a simple definition of recursion?

MULTIPLE CHOICE

Choose the correct answer:

- A. A function that calls itself.
- B. Looping through a 2D list.
- C. Calling many functions consecutively.
- D. Many programmers working together on a task.

3. True or False: Recursion can save programmers a lot of time and resources.

MULTIPLE CHOICE

Choose the correct answer:

- A. True
- B. False

Challenges (2)

1. Books and Sequels

1. You love to read! Some of the best books also have a sequel. You already have a list of page numbers of books and sequels. You decide to write a program that can calculate how many pages total a book and its sequel are from the list you already have.
2. Your program will recursively run through the list and add the pairs together.

Hint: Here's how to create a list of numbers from an input.

```
books = [int(n) for n in input("Input a list of numbers").split()]
```

3. Make sure the inputted list has an even number of data points.

4. Your program will create a function that runs through the list named books. It will check to see if the length of the list is equal to 2. If it is, it will add those data points together and print them.
5. If it isn't, it will find the middle of the list. It will then call the same function on the first half of the list. It will also call the same function on the second half of the list.

For example:

Input: 2 3 10 20 50 50 100 100

Output: 5 30 100 200

2. Pollinating Bees

You are a farmer and have an orchard of fruit trees. It's spring time and it's important that your trees get pollinated so that they can have lots of fruit in the Fall. You don't have time to pollinate all the flowers yourself, so you also keep bees to help with the pollinating. Each flower needs to be visited 3 times to be fully pollinated.

1. Create a recursive program that breaks down the job of pollinating so each bee works on one tree.
2. This program will take a list of numbers as an input.

Hint: This is how to create a list as an input

```
flowers = [int(n) for n in input("How many blossoms are on each tree?").split()]
```

3. The program will then call a function on itself so that each bee helps with one tree. Print out how many flowers total the bee needs to visit in order to have each flower visited 3 times.

For example:

Input: 10 20 15 19 18 30 100

Output: 30 60 45 57 54 90 300

Answer Keys & Solutions

Checkpoint Solutions

Recursion

```
1 this_list = ['panda', 'lion', 'giraffe', 'tiger', 'elephant', 'monkey', 'fish',  
2 'snake', 'bearded dragon', 'koala']  
3 def feeding(this_list):  
4     if len(this_list) == 1:  
5         animal = this_list[0]  
6         print("The " + animal + " has been fed")  
7     else:  
8         mid = len(this_list) // 2  
9         first_half = this_list[:mid]  
10        second_half = this_list[mid:]  
11  
12        feeding(first_half)  
13        feeding(second_half)  
14  
15 feeding(this_list)
```

Questions

1. Why do programmers use recursion when there are other ways to get the same result? Select all that apply.

SELECT MULTIPLE

Correct Answers:

- A. More calculations can be done with less code. ✓ Correct
- B. It saves time and resources. ✓ Correct
- C. It helps break down complicated tasks into easier chunks. ✓ Correct
- D. It enlists the help of more programmers, making their jobs easier. ✗ Incorrect

Explanation:

One recursive function can be made by one programmer.

2. What is a simple definition of recursion?

Correct Answer:

- A. A function that calls itself. ✓ Correct
- B. Looping through a 2D list. ✗ Incorrect
- C. Calling many functions consecutively. ✗ Incorrect
- D. Many programmers working together on a task. ✗ Incorrect

Explanation:

Recursion is when a function can run many times within itself.

3. True or False: Recursion can save programmers a lot of time and resources.

MULTIPLE CHOICE

Correct Answer:

- A. True ✓ Correct
- B. False ✗ Incorrect

Explanation:

Recursion helps to do more computing with less coding.

Challenges

1. Books and Sequels

Solution:

```
1 books = [int(n) for n in input("Input an even list of numbers").split()]
2
3 def pairs(books):
4     if len(books) == 2:
5         print(books[0] + books[1])
6
7     else:
8         mid = len(books) // 2
9         first_half = books[:mid]
10        second_half = books[mid:]
11
12        pairs(first_half)
13        pairs(second_half)
14
15 pairs(books)
```


2. Pollinating Bees

Solution:

```
1 flowers = [int(n) for n in input("How many blossoms are on each tree?").split()]
2
3 def orchard(flowers):
4     if len(flowers) == 1:
5
6         print(flowers[0] * 3)
7     else:
8         mid = len(flowers) // 2
9         first_half = flowers[:mid]
10        second_half = flowers[mid:]
11        orchard(first_half)
12        orchard(second_half)
13 orchard(flowers)
```