



Workshop

Hack the Technical Interview: Algorithms Practice

Presented by Citrix

MLH localhost

CITRIX®

1

*Using your Web Browser,
Open this URL:*

<http://mlhlocal.host/lhd-resources>

2

Click on the workshop you're attending, and find:

- Setup Instructions
- The Code Samples
- A demo project
- A Workshop FAQ
- These Workshop Slides
- More Learning Resources



***Our Mission** is to Empower Hackers.*

65,000+
HACKERS

12,000+
PROJECTS CREATED

3,000+
SCHOOLS

We hope you learn something awesome today!
Find more resources: <http://mlh.io/>

Presented in partnership by



Citrix powers a better way to work by delivering the experience, security and choice that people and organization need to unlock innovation, engage customers, and be productive – anytime, anywhere.

Their Digital Workspaces combine the apps and files needed to get work done, with access control and endpoint management that offer the contextual, secure, unified experiences organization need to do their best work.

Learn about Citrix Career Opportunities!

<http://mlhlocal.host/citrix-careers>

What will you **learn** today?

- 1 Best practices for technical interviews.
- 2 How to solve several common technical interview questions.
- 3 Where to go for more practice.

Table of Contents

- 1. Technical Interview Tips
- 2. Demonstration
- 3. Practice
- 4. Review & Quiz
- 5. Next Steps

Technical Interview Tips

Tip #1: It is OK to ask questions!

Begin by asking parameter questions - they really want to see that you think about what you're doing before you begin coding. For example, if you're asked to reverse a string, ask if there are any built-in methods that you aren't allowed to use (like `reverse()`).



Technical Interview Tips

Tip #2: Plan your answer before you begin coding or whiteboarding.

Say things like "The first thing this function needs to do is ..."
It's ok to write some things down or make notes to yourself! They want to see you planning.



Technical Interview Tips

Tip #3: Start coding or whiteboarding, and talk through parts where you're stuck.

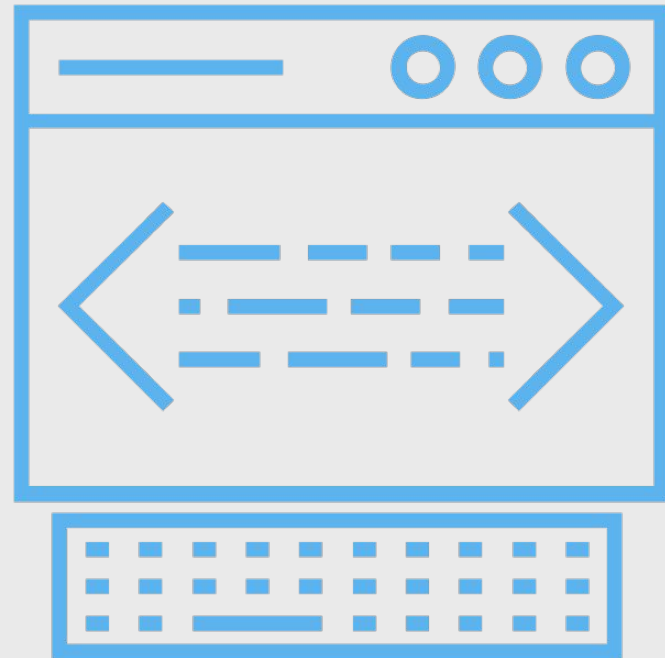
Technical interviewers know what they can and can't share with you. They're very likely to give you a hint if you're close (like if can't remember the name of the method you want to use).



Technical Interview Tips

Tip #4: It's important to refactor!

Once you're satisfied with the way your function works, pause and think about how you can refactor. Remember to talk through what you're doing! Say "I want to refactor this line, because ..."



Technical Interview Tips

Tip #5: Explain your answer thoroughly when you've finished.

Some companies actually place a higher value on your ability to explain how your answer works and why you made the choices that you did than the answer itself. Talk about what your solution does, why you chose this over that, etc.



Technical Interview Tips

Tip #6: Watch the time.

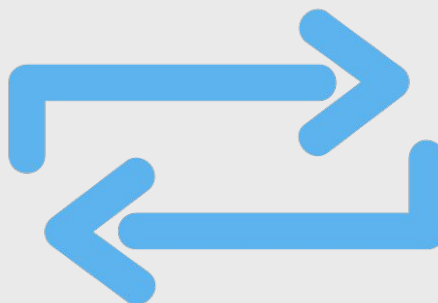
Break the time down. You want to give yourself something like this:

- 10% to plan
- 50% to implement
- 20% to refactor
- 20% to discuss




Technical Interview Tips

To Recap:



1. It is OK to ask questions!
2. Plan your answer before you begin coding or whiteboarding.
3. Talk through the parts where you're stuck.
4. It's important to refactor!
5. Explain your answer thoroughly when you've finished.
6. Watch the time.

Table of Contents

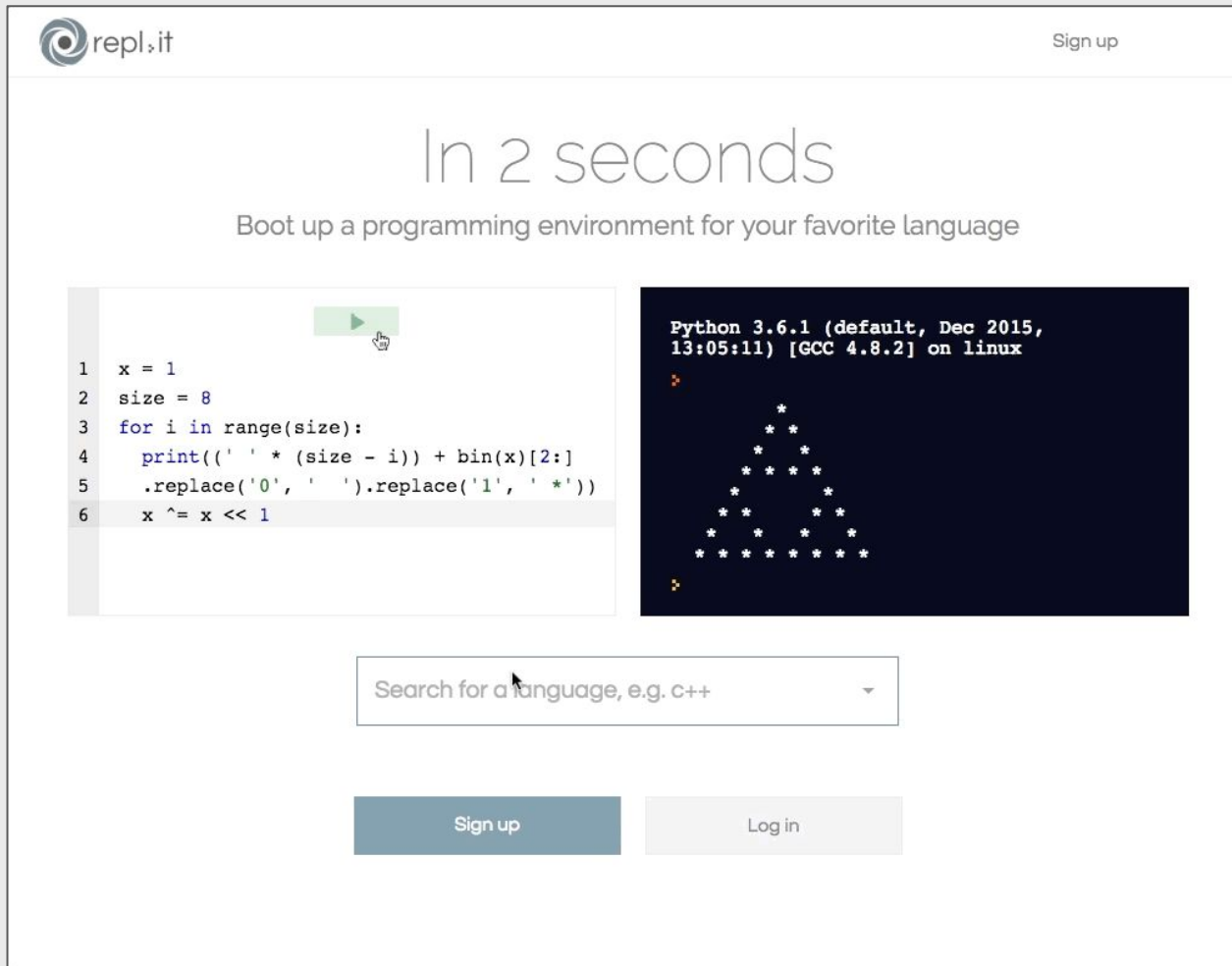
1. Technical Interview Tips
-  2. Demonstration
3. Practice
4. Review & Quiz
5. Next Steps

repl.it

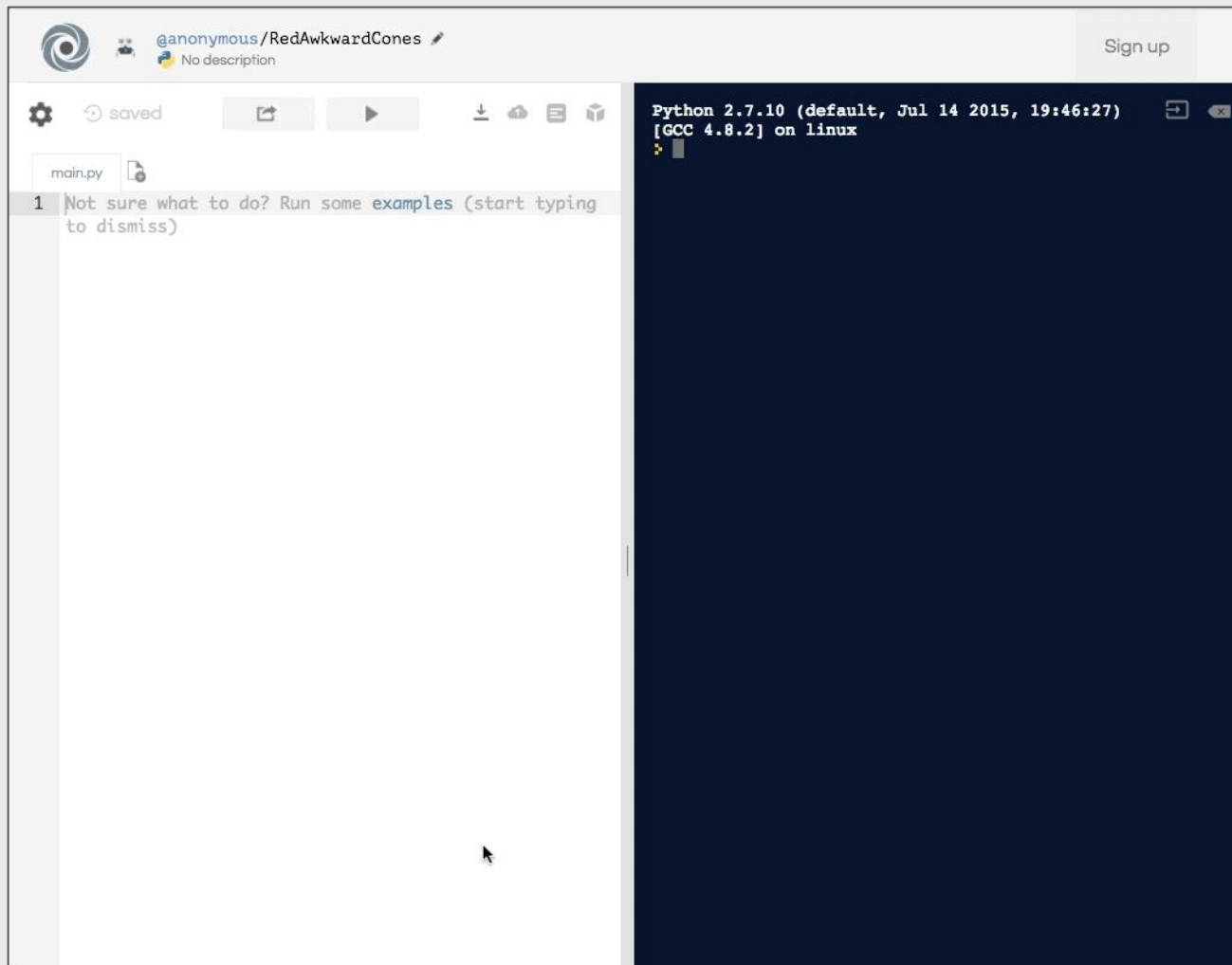
As you work through some of these problems, you might want to check your solutions. While you might not be able to do this in a formal interview, during practice it's a great idea to use a tool like repl.it to debug or share your code!



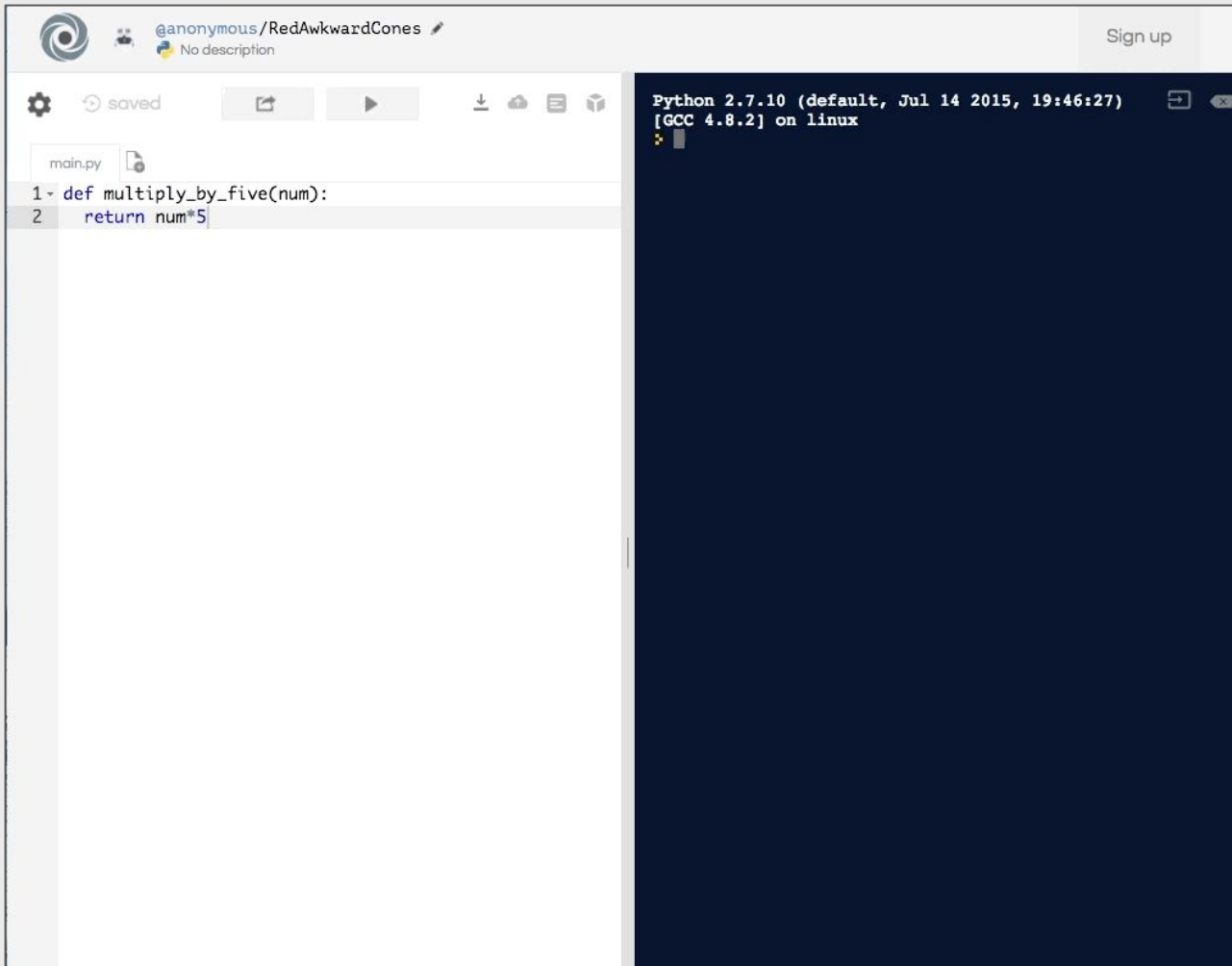
Step 1: On **repl.it**, search for the language you want to use.



Step 2: Type your solution in the code editor on the left.



Step 3: Call your function and click the Run button.



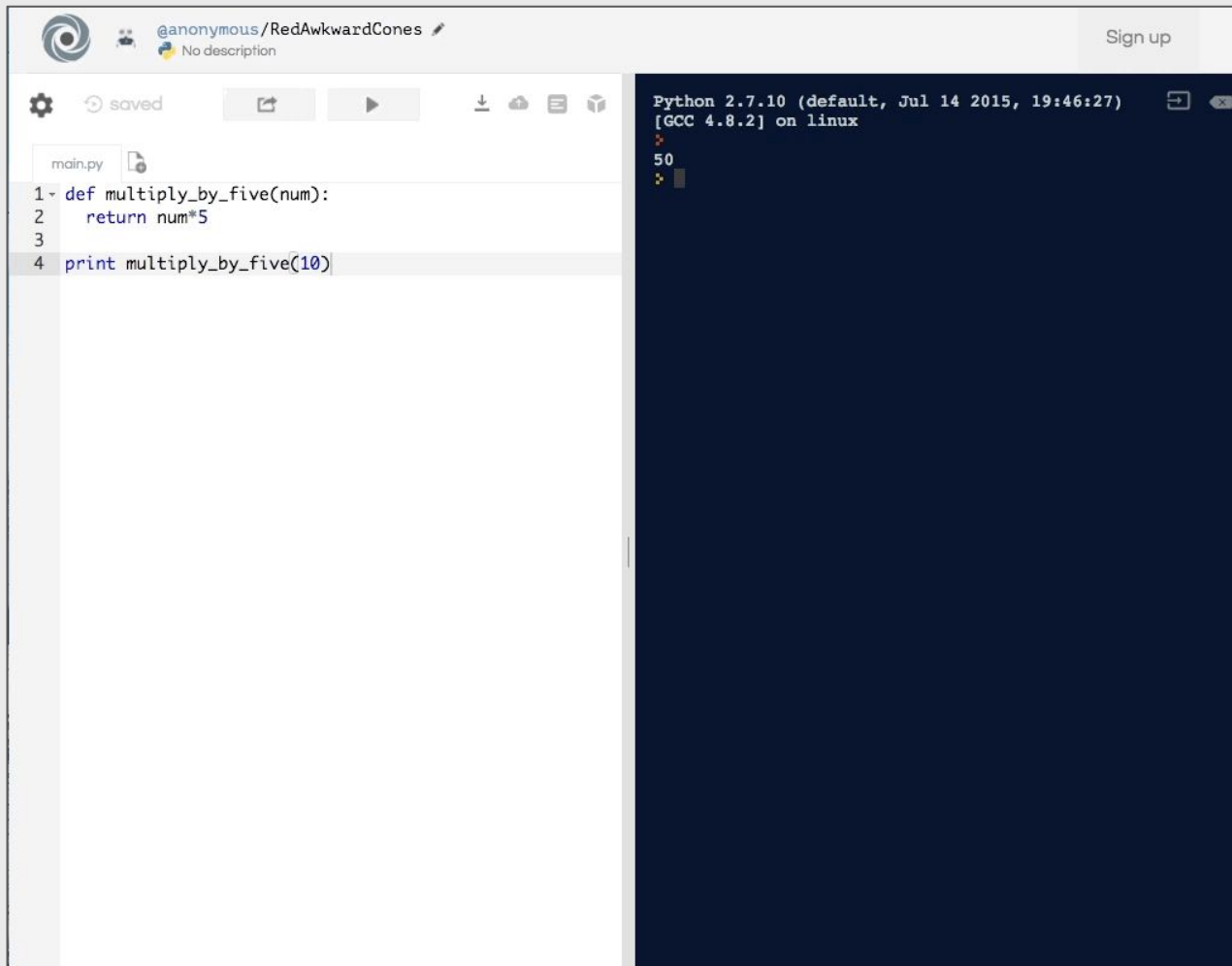
The screenshot shows a web-based code editor interface. At the top, there is a header bar with a logo on the left, a user profile icon and name "@anonymous/RedAwkwardCones" in the center, and a "Sign up" button on the right. Below the header, there is a toolbar with icons for settings, saved status, file operations, and execution. The main area is split into two panes. The left pane shows a file named "main.py" with the following Python code:

```
1 def multiply_by_five(num):  
2     return num*5
```

The right pane is a dark-themed terminal window showing the output of the code execution:

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)  
[GCC 4.8.2] on linux  
✖
```

Step 4: To share a solution later, tell the presenter the name at the end of the URL.



The screenshot shows a web-based Python IDE interface. The top bar includes a logo, the username '@anonymous/RedAwkwardCones', and a 'Sign up' button. Below the bar, there are icons for settings, saved status, and file operations. The main area is split into two panes. The left pane is a code editor showing a file named 'main.py' with the following Python code:

```
1 def multiply_by_five(num):  
2     return num*5  
3  
4 print multiply_by_five(10)
```

The right pane is a terminal window with a dark background. It shows the output of the code execution:

```
Python 2.7.10 (default, Jul 14 2015, 19:46:27)  
[GCC 4.8.2] on linux  
50
```

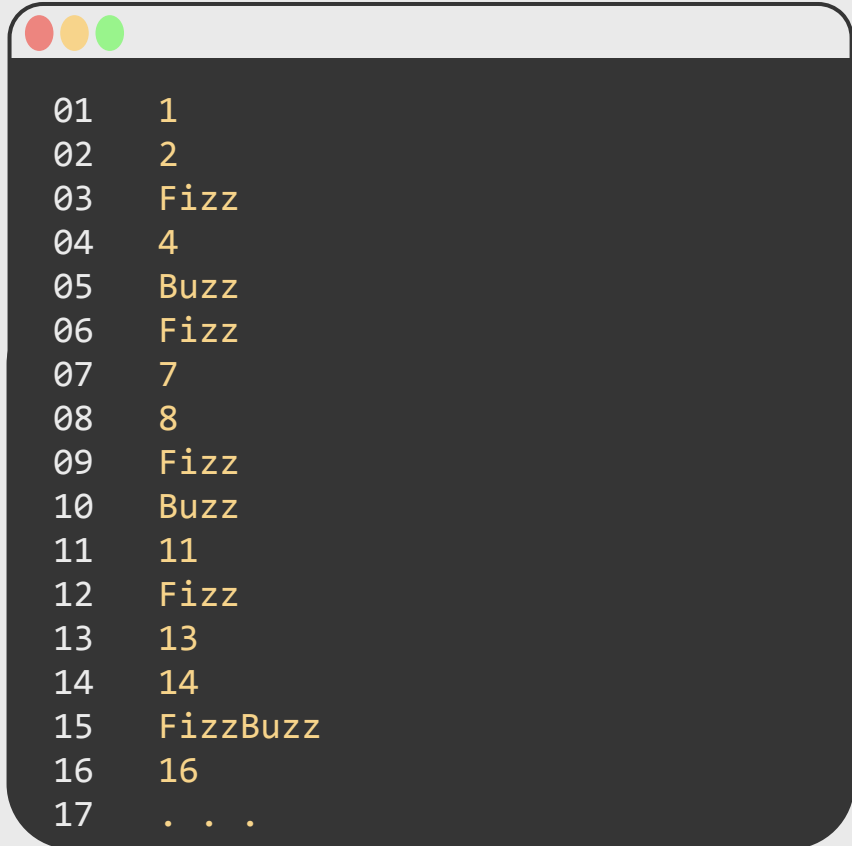
Let's do a Practice Problem!

Problem Statement:

Write a function that takes a single input, a number, and prints out 1 through the number. If the number being printed is divisible by 3, print "Fizz" instead of the number itself. If the number being printed is divisible by 5, print "Buzz." If the number being printed is divisible by 3 AND 5, print "FizzBuzz."

Pro-Tip: Take a picture of this problem to refer back to.

Example Output:

A terminal window with a dark background and light-colored text. It shows the output of a FizzBuzz program for numbers 1 through 17. The output is as follows:

```
01 1
02 2
03 Fizz
04 4
05 Buzz
06 Fizz
07 7
08 8
09 Fizz
10 Buzz
11 11
12 Fizz
13 13
14 14
15 FizzBuzz
16 16
17 . . .
```

Tip #1: Ask questions!

What are some questions you might ask before solving this problem? Share example questions.



Tip #2: Plan your answer.

This is called pseudocode. We wrote out exactly what we want the function to do in plain language. Later we can "translate" these steps into the coding language of our choice.

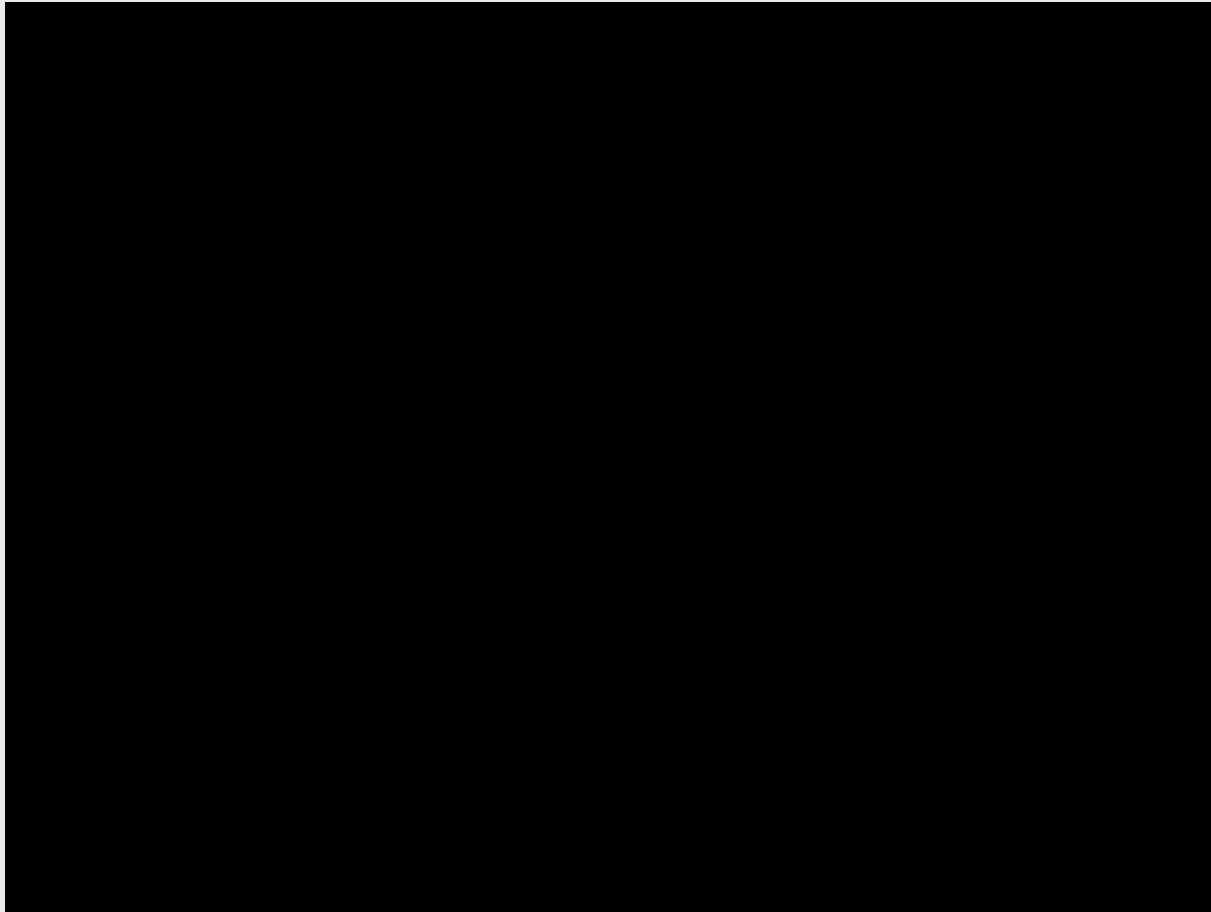
```
01  ## function should take one parameter
02
03  ## then check if the element is divisible by 3
04  ## if yes, print "Fizz"
05
06  ## otherwise, check if the element is divisible by 5
07  ## if yes, print "Buzz"
08
09  ## otherwise, check if the element is divisible by both 3 && 5
10  ## if yes, print "FizzBuzz"
11
12  ## otherwise, print the number itself.
```

Tip #3: Write out your solution.

This solution translates the pseudocode from the last slide into a function, step-by-step. This solution is in Python, but the language doesn't matter.

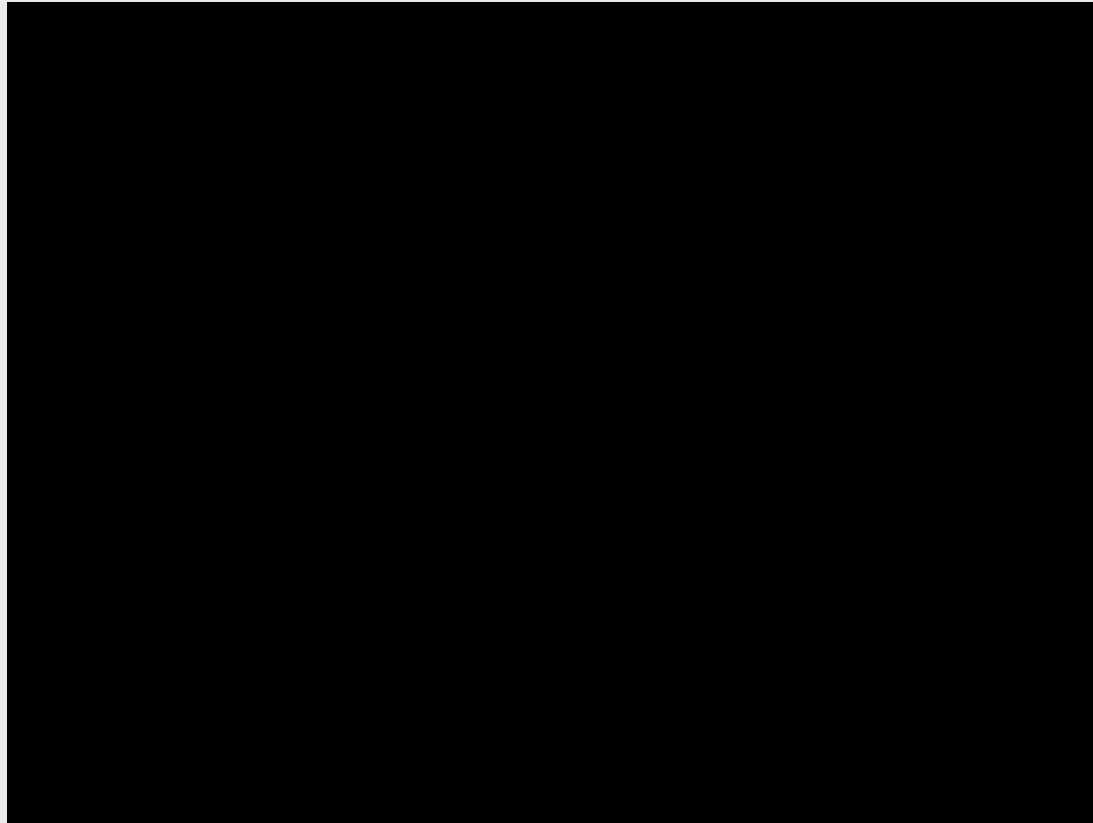
```
01  def fizz_buzz(num):  
02  
03      for x in range(1, num+1):  
04          if x % 3 == 0:  
05              print('Fizz')  
06          elif x % 5 == 0:  
07              print('Buzz')  
08          elif x % 3 == 0 and x % 5 == 0:  
09              print('FizzBuzz')  
10          else:  
11              print(x)  
12
```

Let's check our solution!



Tip #4: It's important to refactor.

Did you see what went wrong in the video on the previous slide?
This is why you talk through your code and refactor!



Tip #5: Explain your answer.

```
01  def fizz_buzz(num):  
02  
03      for x in range(1, num+1):  
04          if x % 3 == 0 and x % 5 == 0:  
05              print('FizzBuzz')  
06          elif x % 3 == 0:  
07              print('Fizz')  
08          elif x % 5 == 0:  
09              print('Buzz')  
10          else:  
11              print(x)  
12
```

Line 1 defines a function with a single parameter.

Tip #5: Explain your answer.

```
01  def fizz_buzz(num):  
02  
03      for x in range(1, num+1):  
04          if x % 3 == 0 and x % 5 == 0:  
05              print('FizzBuzz')  
06          elif x % 3 == 0:  
07              print('Fizz')  
08          elif x % 5 == 0:  
09              print('Buzz')  
10          else:  
11              print(x)  
12
```

Line 3 creates a loop that will iterate from 1 through the number passed through the function.

Tip #5: Explain your answer.

```
01 def fizz_buzz(num):  
02  
03     for x in range(1, num+1):  
04         if x % 3 == 0 and x % 5 == 0:  
05             print('FizzBuzz')  
06         elif x % 3 == 0:  
07             print('Fizz')  
08         elif x % 5 == 0:  
09             print('Buzz')  
10         else:  
11             print(x)  
12
```

Line 4 checks if the current value is divisible by both 3 and 5.

Line 5 prints "FizzBuzz" if that's true.

Tip #5: Explain your answer.

```
01  def fizz_buzz(num):  
02  
03      for x in range(1, num+1):  
04          if x % 3 == 0 and x % 5 == 0:  
05              print('FizzBuzz')  
06          elif x % 3 == 0:  
07              print('Fizz')  
08          elif x % 5 == 0:  
09              print('Buzz')  
10          else:  
11              print(x)  
12
```

Line 6 checks if the current value is divisible by only 3.
Line 7 prints "Fizz" if that's true.

Tip #5: Explain your answer.

```
01  def fizz_buzz(num):  
02  
03      for x in range(1, num+1):  
04          if x % 3 == 0 and x % 5 == 0:  
05              print('FizzBuzz')  
06          elif x % 3 == 0:  
07              print('Fizz')  
08          elif x % 5 == 0:  
09              print('Buzz')  
10          else:  
11              print(x)  
12
```

Line 8 checks if the current value is divisible by only 5. Line 9 prints "Buzz" if that's true.

Tip #5: Explain your answer.

```
01 def fizz_buzz(num):  
02  
03     for x in range(1, num+1):  
04         if x % 3 == 0 and x % 5 == 0:  
05             print('FizzBuzz')  
06         elif x % 3 == 0:  
07             print('Fizz')  
08         elif x % 5 == 0:  
09             print('Buzz')  
10         else:  
11             print(x)  
12
```

Line 12 prints the number itself if none of the previous conditions were met.

Tip #6: Watch the time!

How long did that take? Do you think we split up our time efficiently?
Did we hit all the steps?

Remember, you want to give yourself something like this:

- 10% to plan
- 50% to implement
- 20% to refactor
- 20% to discuss

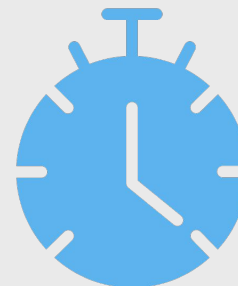


Table of Contents

1. Technical Interview Tips

2. Demonstration



3. Practice

4. Review & Quiz

5. Next Steps

How This Works

1. Make sure you have a partner!
2. Two problems will be presented. Each problem has an Interviewer Script. One person should be the interviewer and the other person should be the candidate. Switch places for the second problem.
3. The URL to find the Interview Script is on the slide with the problem statement. The interviewer should pull it up on their phone or laptop.

How This Works

4. Once the problem is up on the board, do your best to treat it like a real interview! Follow the tips we've covered so far.
5. When time is up, we'll review some solutions together on the board.
6. Be brave! Share your solutions.



String Rotation

Write a function that takes in two string inputs, and returns true if they are a rotation of each other.

Time: 35 minutes

```
01  ## input
02  rotate("ABCD", "BCDA")
03
04  ## output
05  TRUE
06
07  ## input
08  rotate("ABCD", "ACDB")
09
10  ## output
11  FALSE
12
```

Time's Up!

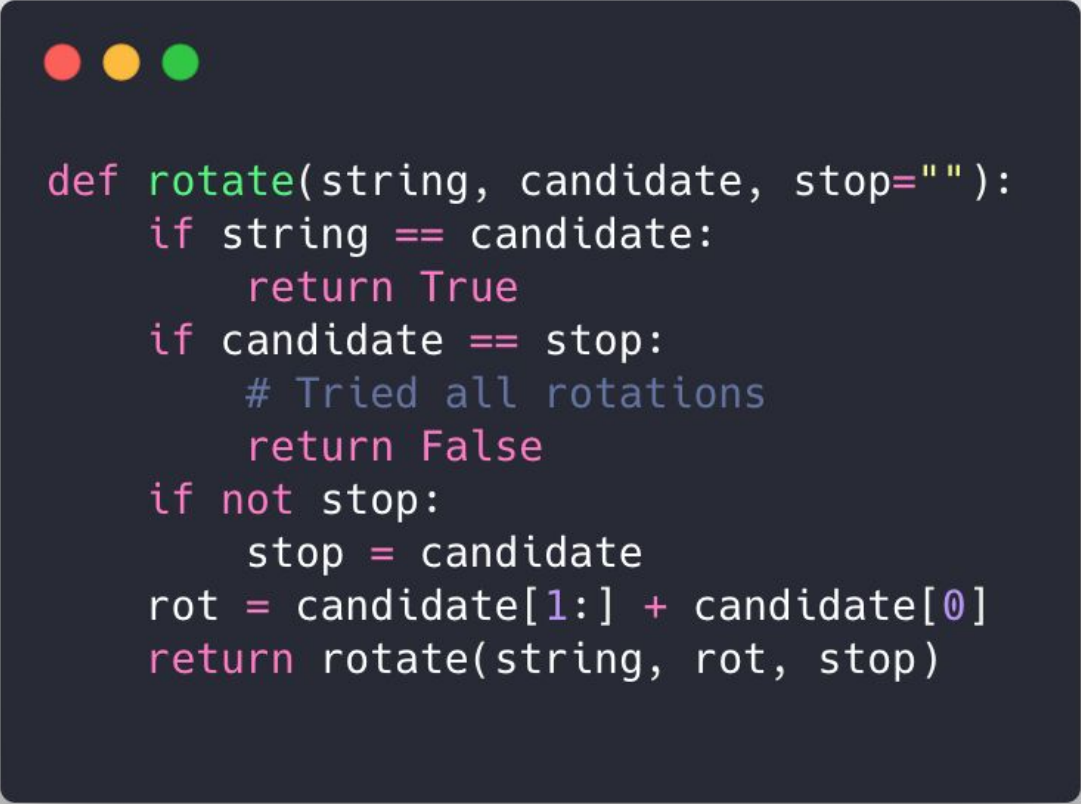
Does anyone want to share their solution?

Share your repl.it URL and then we'll discuss!

Max time: 5 minutes



Solution #1



```
def rotate(string, candidate, stop=""):  
    if string == candidate:  
        return True  
    if candidate == stop:  
        # Tried all rotations  
        return False  
    if not stop:  
        stop = candidate  
    rot = candidate[1:] + candidate[0]  
    return rotate(string, rot, stop)
```

Solution #1

```
def rotate(string, candidate, stop=""):
    if string == candidate:
        return True
    if candidate == stop:
        # Tried all rotations
        return False
    if not stop:
        stop = candidate
    rot = candidate[1:] + candidate[0]
    return rotate(string, rot, stop)
```

Line 1 - Check if the candidate is the same as the string, if so then this is valid

Solution #1

```
def rotate(string, candidate, stop=""):
    if string == candidate:
        return True
    if candidate == stop:
        # Tried all rotations
        return False
    if not stop:
        stop = candidate
    rot = candidate[1:] + candidate[0]
    return rotate(string, rot, stop)
```

Line 3 - Stop means that we've rotated the whole string on it self, we have no more combinations

Solution #1

```
def rotate(string, candidate, stop=""):
    if string == candidate:
        return True
    if candidate == stop:
        # Tried all rotations
        return False
    if not stop:
        stop = candidate
    rot = candidate[1:] + candidate[0]
    return rotate(string, rot, stop)
```

Line 6 - If stop is empty, then set it to be the candidate (so we know when to stop)

Solution #1

```
def rotate(string, candidate, stop=""):
    if string == candidate:
        return True
    if candidate == stop:
        # Tried all rotations
        return False
    if not stop:
        stop = candidate
    rot = candidate[1:] + candidate[0]
    return rotate(string, rot, stop)
```

Line 8 - Begin rotating the string using recursion.

Solution #2



```
def rotate(original, candidate):  
    return len(original) == len(candidate) and candidate in original*2
```

Solution #2



```
def rotate(original, candidate):  
    return len(original) == len(candidate) and candidate in original*2
```

First to shortcut the algorithm we ensure the length are the same.

Next, we double the original string and check whether the candidate string is inside. Why do we need to double it?

Why is Solution #2 Better?

- Solution 2 is much more simple (only one line!)
- Solution 2 does not require recursion
- Solution 1 is *very* prone to human error
- Solution 2 leverages Python's built in API

Zig-Zag Arrays

Write a function that takes an array with distinct elements and sorts them in a zig-zag fashion. (ie

$a < b > c < d > e < f$)

Time: 35 minutes

```
01  ## input
02  zigzag([4, 3, 7, 8, 6, 2, 1])
03
04  ## output (can be different order)
05  [3, 7, 4, 8, 2, 6, 1]
06
07  ## input
08  zigzag([1, 4, 3, 2])
09
10  ## output (can be different order)
11  [1, 4, 2, 3]
12
```

Time's Up!

Does anyone want to share their solution?

Share your repl.it URL and then we'll discuss!

Max time: 5 minutes



Solution #1



```
def swap(arr,i,j):  
    arr[i],arr[j] = arr[j],arr[i]  
  
def zigzag(arr):  
    srt = sorted(arr)  
    left = 1  
    while left < len(srt)-1:  
        swap(srt, left, left + 1)  
        left = left + 2  
    print(srt)
```


Solution #1

```
def swap(arr,i,j):  
    arr[i],arr[j] = arr[j],arr[i]  
  
def zigzag(arr):  
    srt = sorted(arr)  
    left = 1  
    while left < len(srt)-1:  
        swap(srt, left, left + 1)  
        left = left + 2  
    print(srt)
```

Swap function is a helper we can use to swap array elements

Solution #1

```
def swap(arr,i,j):  
    arr[i],arr[j] = arr[j],arr[i]  
  
def zigzag(arr):  
    srt = sorted(arr)  
    left = 1  
    while left < len(srt)-1:  
        swap(srt, left, left + 1)  
        left = left + 2  
    print(srt)
```

Line 5 - is where we sort the array. We can use any sorting algorithm but assume $n \log n$ complexity.

Solution #1



```
def swap(arr,i,j):  
    arr[i],arr[j] = arr[j],arr[i]  
  
def zigzag(arr):  
    srt = sorted(arr)  
    left = 1  
    while left < len(srt)-1:  
        swap(srt, left, left + 1)  
        left = left + 2  
    print(srt)
```

Line 6-9 - Using a while loop, we can iterate through the sorted array started at index 1, we swap higher to lower and skip by two. This ensures that we have zig-zag arrays.

Solution #1



```
def swap(arr,i,j):  
    arr[i],arr[j] = arr[j],arr[i]  
  
def zigzag(arr):  
    srt = sorted(arr)  
    left = 1  
    while left < len(srt)-1:  
        swap(srt, left, left + 1)  
        left = left + 2  
    print(srt)
```

This is a very simple approach
but what is the complexity?

($n \log n$ because of sorting)

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

Line 2: Flag true indicates relation "<" is expected, else ">" is expected. The first expected relation is "<"

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

The for-loop iterates through the array, but what does it do?

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

The for loop checks the flag and sees what relation is needed.

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

Line 5 checks to see if the number at position $i + 1$ is not the intended flag, if it that is true then swap the numbers.

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

Line 8 does the same thing as Line 5 but for the opposite relation

Solution #2

```
def zigzag(arr, n):  
    flag = True  
    for i in range(n-1):  
        if flag is True:  
            if arr[i] > arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        else:  
            if arr[i] < arr[i+1]:  
                arr[i],arr[i+1] = arr[i+1],arr[i]  
        flag = bool(1 - flag)  
    print(arr)  
  
def zigzag(arr):  
    zigzag(arr, len(arr))
```

Line 10 flips the relation so that we can zig-zag.

Why is Solution #2 Better?

- Solution 2 does not require $O(n \log n)$ sorting
- Solution 2 iterates through the array only once
- Solution 2 has complexity of $O(n)$
- Solution using $O(1)$ memory (auxiliary space)

Pythagorean Triplet

Write a function that takes an array of elements and returns "Yes" if there is a pythagorean triplet ((a, b, c) that satisfies $a^2 + b^2 = c^2$), "No" otherwise.

Time: 35 minutes

```
01  ## input
02  triplet([3, 1, 4, 6, 5])
03
04  ## output
05  Yes
06
07  ## input
08  triplet([10, 4, 6, 12, 5])
09
10  ## output
11  No
12
```

Time's Up!

Does anyone want to share their solution?

Share your repl.it URL and then we'll discuss!

Max time: 5 minutes



Solution #1

```
def isTriplet(ar, n):  
    j=0  
  
    for i in range(n - 2):  
        for k in range(j + 1, n):  
            for j in range(i + 1, n - 1):  
                x = ar[i]*ar[i]  
                y = ar[j]*ar[j]  
                z = ar[k]*ar[k]  
                if (x == y + z or y == x + z or z == x + y):  
                    return 1  
  
    return 0  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Solution #1

```
def isTriplet(ar, n):  
    j=0  
  
    for i in range(n - 2):  
        for k in range(j + 1, n):  
            for j in range(i + 1, n - 1):  
                x = ar[i]*ar[i]  
                y = ar[j]*ar[j]  
                z = ar[k]*ar[k]  
                if (x == y + z or y == x + z or z == x + y):  
                    return 1  
  
    return 0  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Naive method - create 3 for loops and try all the possible combinations

Solution #1

```
def isTriplet(ar, n):  
    j=0  
  
    for i in range(n - 2):  
        for k in range(j + 1, n):  
            for j in range(i + 1, n - 1):  
                x = ar[i]*ar[i]  
                y = ar[j]*ar[j]  
                z = ar[k]*ar[k]  
                if (x == y + z or y == x + z or z == x + y):  
                    return 1  
  
    return 0  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 7-9 - is where we calculate the square of each number

Solution #1

```
def isTriplet(ar, n):  
    j=0  
  
    for i in range(n - 2):  
        for k in range(j + 1, n):  
            for j in range(i + 1, n - 1):  
                x = ar[i]*ar[i]  
                y = ar[j]*ar[j]  
                z = ar[k]*ar[k]  
                if (x == y + z or y == x + z or z == x + y):  
                    return 1  
  
    return 0  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 10 - is where we check whether the pythagorean trip

Solution #1

```
def isTriplet(ar, n):  
    j=0  
  
    for i in range(n - 2):  
        for k in range(j + 1, n):  
            for j in range(i + 1, n - 1):  
                x = ar[i]*ar[i]  
                y = ar[j]*ar[j]  
                z = ar[k]*ar[k]  
                if (x == y + z or y == x + z or z == x + y):  
                    return 1  
  
    return 0  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 11 - if we reached here then we couldn't find a pythagorean triplet

Solution #1

```
def isTriplet(ar, n):  
    j=0  
  
    for i in range(n - 2):  
        for k in range(j + 1, n):  
            for j in range(i + 1, n - 1):  
                x = ar[i]*ar[i]  
                y = ar[j]*ar[j]  
                z = ar[k]*ar[k]  
                if (x == y + z or y == x + z or z == x + y):  
                    return 1  
  
    return 0  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

What is the complexity?

$O(n^3)$

Can we do better?

Solution #2

```
def isTriplet(ar, n):  
    for i in range(n):  
        ar[i] = ar[i] * ar[i]  
    ar.sort()  
    for i in range(n-1, 1, -1):  
        j = 0  
        k = i - 1  
        while (j < k):  
            if (ar[j] + ar[k] == ar[i]):  
                return True  
            else:  
                if (ar[j] + ar[k] < ar[i]):  
                    j = j + 1  
                else:  
                    k = k - 1  
    return False  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Solution #2

```
def isTriplet(ar, n):
    for i in range(n):
        ar[i] = ar[i] * ar[i]
    ar.sort()
    for i in range(n-1, 1, -1):
        j = 0
        k = i - 1
        while (j < k):
            if (ar[j] + ar[k] == ar[i]):
                return True
            else:
                if (ar[j] + ar[k] < ar[i]):
                    j = j + 1
                else:
                    k = k - 1
    return False

def triplet(ar):
    if(isTriplet(ar, len(ar))):
        print("Yes")
    else:
        print("No")
```

Steps:

1. Square every element
2. Sort the squared array
3. Start with last element, and try to find two numbers in the array that add up to this one.
4. If none, move one index back, try again.

Solution #2

```
def isTriplet(ar, n):  
    for i in range(n):  
        ar[i] = ar[i] * ar[i]  
    ar.sort()  
    for i in range(n-1, 1, -1):  
        j = 0  
        k = i - 1  
        while (j < k):  
            if (ar[j] + ar[k] == ar[i]):  
                return True  
            else:  
                if (ar[j] + ar[k] < ar[i]):  
                    j = j + 1  
                else:  
                    k = k - 1  
    return False  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 2 - is where we generate new array of squared numbers

Solution #2

```
def isTriplet(ar, n):  
    for i in range(n):  
        ar[i] = ar[i] * ar[i]  
    ar.sort()  
    for i in range(n-1, 1, -1):  
        j = 0  
        k = i - 1  
        while (j < k):  
            if (ar[j] + ar[k] == ar[i]):  
                return True  
            else:  
                if (ar[j] + ar[k] < ar[i]):  
                    j = j + 1  
                else:  
                    k = k - 1  
    return False  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 3 - sort the squared array

Solution #2

```
def isTriplet(ar, n):  
    for i in range(n):  
        ar[i] = ar[i] * ar[i]  
    ar.sort()  
    for i in range(n-1, 1, -1):  
        j = 0  
        k = i - 1  
        while (j < k):  
            if (ar[j] + ar[k] == ar[i]):  
                return True  
            else:  
                if (ar[j] + ar[k] < ar[i]):  
                    j = j + 1  
                else:  
                    k = k - 1  
    return False  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 4 - Start at the last element and go backwards in the array until index 2

Solution #2

```
def isTriplet(ar, n):
    for i in range(n):
        ar[i] = ar[i] * ar[i]
    ar.sort()
    for i in range(n-1, 1, -1):
        j = 0
        k = i - 1
        while (j < k):
            if (ar[j] + ar[k] == ar[i]):
                return True
            else:
                if (ar[j] + ar[k] < ar[i]):
                    j = j + 1
                else:
                    k = k - 1
    return False

def triplet(ar):
    if(isTriplet(ar, len(ar))):
        print("Yes")
    else:
        print("No")
```

Line 5-6 - Select two numbers at the corner of the constrained array and start checking if they add up to number at index i.

Solution #2

```
def isTriplet(ar, n):  
    for i in range(n):  
        ar[i] = ar[i] * ar[i]  
    ar.sort()  
    for i in range(n-1, 1, -1):  
        j = 0  
        k = i - 1  
        while (j < k):  
            if (ar[j] + ar[k] == ar[i]):  
                return True  
            else:  
                if (ar[j] + ar[k] < ar[i]):  
                    j = j + 1  
                else:  
                    k = k - 1  
    return False  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 8 - A triplet is found

Solution #2

```
def isTriplet(ar, n):
    for i in range(n):
        ar[i] = ar[i] * ar[i]
    ar.sort()
    for i in range(n-1, 1, -1):
        j = 0
        k = i - 1
        while (j < k):
            if (ar[j] + ar[k] == ar[i]):
                return True
            else:
                if (ar[j] + ar[k] < ar[i]):
                    j = j + 1
                else:
                    k = k - 1
    return False

def triplet(ar):
    if(isTriplet(ar, len(ar))):
        print("Yes")
    else:
        print("No")
```

Line 10 - if a triplet is not found, check if the addition of the two squared numbers is less than the one at index i , then move the left index up one.

Solution #2

```
def isTriplet(ar, n):  
    for i in range(n):  
        ar[i] = ar[i] * ar[i]  
    ar.sort()  
    for i in range(n-1, 1, -1):  
        j = 0  
        k = i - 1  
        while (j < k):  
            if (ar[j] + ar[k] == ar[i]):  
                return True  
            else:  
                if (ar[j] + ar[k] < ar[i]):  
                    j = j + 1  
                else:  
                    k = k - 1  
    return False  
  
def triplet(ar):  
    if(isTriplet(ar, len(ar))):  
        print("Yes")  
    else:  
        print("No")
```

Line 16 - if we reached here then a triplet could not be found.

Why is Solution #2 Better?

- Solution 2 has complexity of $O(n^2)$ compared to $O(n^3)$ for Solution 1
- On average, Solution 2 finds triplets faster than Solution 1 since it leverages sorting

Table of Contents

1. Technical Interview Tips

2. Demonstration

3. Practice

 4. Review & Quiz

5. Next Steps

What did you **learn** today?

- 1 It's good to ask questions!
- 2 Plan your code in advance!
- 3 Your explanation is just as important as your solution.

What did you learn today?

We created a fun quiz to test your knowledge and see what you learned from this workshop.

<http://mlhlocal.host/quiz>

Table of Contents

1. Technical Interview Tips

2. Demonstration

3. Practice

4. Review & Quiz



5. Next Steps

Where to go from here

- 1 Check your email tomorrow for instructions on how to contribute your solution!
- 2 Check out mlhlocal.host/interview-cake for more practice problems.
- 3 Learn more about efficiency and Big O Notation at <http://mlhlocal.host/big-o-notation>



Sign up for the MLH Career Lab!

<http://mlhlocal.host/career-lab>

- Browse a curated list of cool companies to work for.
- Apply for jobs and internships from companies that want to recruit directly from the MLH community.
- Receive updates and career advice from MLH!



Drop your resume with



By attending this workshop you have the opportunity to connect with a Citrix recruiter directly! Click on the button below to join the Citrix talent network.

Submit Resume

Additional Practice Problem #1

Anagrams

Write a function that takes two inputs, a pair of strings, and proves if they are anagrams (contain all the same letters) of each other.

```
01  ## input
02  anagram("dairy", "diary")
03
04  ## output
05  TRUE
06
07  ## input
08  anagram("cars", "face")
09
10  ## output
11  FALSE
12
```

Additional Practice Problem #2

Vowel Counts

Write a function that takes a single input, a string, and return the counts of all the vowels.

```
01  ## input
02  vowels("HELLO, WORLD!")
03
04  ## output
05  { E: 1, O: 2 }
06
07  ## input
08  vowels("Major League Hacking")
09
10  ## output
11  { a: 3, e: 2, i: 1, o: 1, u: 1 }
12
```

Additional Practice Problem #3

Palindromes

Write a function that takes a single input, a string, and return if the string is a palindrome (reads the same backwards as forwards)

```
01  ## input
02  palindrome("racecar")
03
04  ## output
05  TRUE
06
07  ## input
08  palindrome("hacking")
09
10  ## output
11  FALSE
12
```


Additional Practice Problem #4

Valid Parentheses

Write a function that takes a single input, a string, and return if the string has valid parenthesis (open brackets closed with closing brackets). Use '['{' brackets.

```
01  ## input
02  brackets("[[(M){L}[H]]]")
03
04  ## output
05  TRUE
06
07  ## input
08  brackets("(M[L]){h}")
09
10  ## output
11  FALSE
12
```

Learning shouldn't stop when the workshop ends...

Check your email for access to:



- These workshop slides
- Practice problems to keep learning
- Deeper dives into key topics
- Instructions to join the community
- More opportunities from MLH!

Workshop

Hack the Technical Interview: Algorithms Practice

Presented by Citrix

MLH localhost

CITRIX®