



50 Python Interview Questions and Answers

Apr 08, 2020 - 33 min read

Cameron Wilson



Hands-on practice is the best way to prepare for your coding interviews. Thankfully, most recruiters will test the same topics: general language knowledge, data structures, Big O, and sorting.

Today, we'll help you prepare for your next Python interview with 50 of the most asked interview questions.

Here's what we'll cover today:

- Python Language Basics
- Python Coding Interview Questions
- Next Steps

Learn the skills to tackle any interview question.

Refine your interview skills with hands-on projects and practice problems

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) to learn more.

Got it!

Decode the Coding Interview in Python: Real-World Examples

(<https://www.educative.io/courses/decode-coding-interview-python>)

Python Language Basics Questions

Question 1: What is the difference between a list and a tuple?

List	Tuple
A list consists of mutable objects. (Objects which can be changed after creation)	A tuple consists of immutable objects. (Objects which cannot change after creation)
List has a large memory.	Tuple has a small memory.
List is stored in two blocks of memory (One is fixed sized and the other is variable sized for storing data)	Tuple is stored in a single block of memory.
Creating a list is slower because two memory blocks need to be accessed.	Creating a tuple is faster than creating a list.
An element in a list can be removed or replaced.	An element in a tuple cannot be removed or replaced.
A list has data stored in [] brackets. For example, [1,2,3]	A tuple has data stored in () brackets. For example, (1,2,3)

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

When to use each:

A **tuple** should be used whenever the user is aware of what is inserted in the tuple. Suppose that a college stores the information of its students in a data structure; in order for this information to remain immutable it should be stored in a tuple.

Since **lists** provide users with easier accessibility, they should be used whenever similar types of objects need to be stored. For instance, if a grocery needs to store all the dairy products in one field, it should use a list.

Question 2: How would you convert a list into a tuple?

```
my_list = [50, "Twenty", 110, "Fifty", "Ten", 20, 10, 80, "Eighty"]

my_tuple = (my_list[0], my_list[len(my_list) - 1], len(my_list))
print(my_tuple)
```



All we have to do is create a tuple with three elements. The first element of the tuple is the first element of the list, which can be found using `my_list[0]`.

The second element of the tuple is the last element in the list.

`my_list[len(my_list) - 1]` will give us this element. We could also have used the `pop()` method, but that would alter the list.

Question 3: What is the difference between an array and a list?

List

Array

Python lists are very flexible

Python arrays are just a thin

and can hold arbitrary data

wrapper on C arrays

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) to learn more.

Got it!

List	Array
Lists are a part of Python's syntax, so they do not need to be declared first.	Arrays need to first be imported, or declared, from other libraries (i.e. numpy).
Lists can also be re-sized quickly in a time-efficient manner. This is because Python initializes some extra elements in the list at initialization.	Arrays cannot be resized. Instead, an array's values would need to be copied to another larger array.
Lists can hold heterogeneous data.	Arrays can only store homogenous data. They have values with uniform data types.
Mathematical functions cannot be directly applied to lists. Instead, they have to be individually applied to each element.	Arrays are specially optimized for arithmetic computations.
Lists consume more memory as they are allocated a few extra elements to allow for quicker appending of items.	Since arrays stay the size that they were when they were first initialized, they are compact.




Question 4: How would you convert a list to an array?

During programming, there will be instances when you will need to convert existing lists to arrays in order to perform certain operations on them (arrays enable mathematical operations to be performed on them in ways that lists do not).

We use cookies to ensure you have the best browsing experience on our website. [Privacy Policy](#)

Here we'll be using `numpy.array()`. This function of the `numpy` library takes a list as an argument and returns an array that contains all the elements of the list. See the example below:

```
1 import numpy as np
2 my_list = [2,4,6,8,10]
3 my_array = np.array(my_list)
4 # printing my_array
5 print my_array
6 # printing the type of my_array
7 print type(my_array)
```



Question 5: How is memory managed in Python?

1. Memory management in python is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap. The python interpreter takes care of this instead.
2. The allocation of heap space for Python objects is done by Python's memory manager. The core API gives access to some tools for the programmer to code.
3. Python also has an inbuilt garbage collector, which recycles all the unused memory and so that it can be made available to the heap space.

Question 6: How do you achieve multithreading in Python?

1. Python has a multi-threading package but if you want to multi-thread to speed your code up, then it's usually not a good idea to use it.
2. Python has a construct called the Global Interpreter Lock (GIL). The GIL makes sure that only one of your 'threads' can execute at any one time. A thread acquires the GIL, does a little work, then passes the GIL onto the next thread.
3. This happens very quickly so to the human eye it may seem like your threads are executing in parallel, but they are really just taking turns using the same CPU core.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

4. All this GIL passing adds overhead to execution. This means that if you want to make your code run faster then using the threading package often isn't a good idea.

Got it!

Question 7: What is monkey patching?

In Python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

Question 8: What is a lambda function? Give an example of when it's useful and when it's not.

A lambda function is a small anonymous function, which returns an object.

The object returned by lambda is usually assigned to a variable or used as a part of other bigger functions.

Instead of the conventional `def` keyword used for creating functions, a lambda function is defined by using the `lambda` keyword.

The purpose of lambdas

A lambda is much more readable than a full function since it can be written in-line. Hence, it is a good practice to use lambdas when the function expression is small.

The beauty of lambda functions lies in the fact that they return function objects. This makes them helpful when used with functions like `map` or `filter` which require function objects as arguments.

Lambdas aren't useful when the expression exceeds a single line.

Question 9: What is pickling and unpickling?

Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using `dump` function, this process is called pickling. While the process of retrieving original

Python objects from the stored string representation is called unpickling.

Got it!

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) to learn more.

Question 10: What advantages do NumPy arrays offer over (nested) Python lists?

1. Python's lists are efficient general-purpose containers. They support (fairly) efficient insertion, deletion, appending, and concatenation, and Python's list comprehensions make them easy to construct and manipulate.
2. They have certain limitations: they don't support "vectorized" operations like elementwise addition and multiplication, and the fact that they can contain objects of differing types mean that Python must store type information for every element, and must execute type dispatching code when operating on each element.
3. NumPy is not just more efficient; it is also more convenient. You get a lot of vector and matrix operations for free, which sometimes allow one to avoid unnecessary work. And they are also efficiently implemented.
4. NumPy array is faster and You get a lot built in with NumPy, FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, etc.

Question 11: Explain inheritance in Python with an example

Inheritance allows One class to gain all the members(say attributes and methods) of another class. Inheritance provides code reusability, makes it easier to create and maintain an application. The class from which we are inheriting is called super-class and the class that is inherited is called a derived / child class.

They are different types of inheritance supported by Python:

1. Single Inheritance – where a derived class acquires the members of a single super class.

2. Multi-level inheritance – a derived class d1 is inherited from base class base1, and d2 are inherited from base2.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

3. Hierarchical inheritance – from one base class you can inherit any number of child classes
4. Multiple inheritance – a derived class is inherited from more than one base class.

Question 12: What is polymorphism in Python?

Polymorphism means the ability to take multiple forms. So, for instance, if the parent class has a method named ABC then the child class also can have a method with the same name ABC having its own parameters and variables. Python allows for polymorphism.

Question 13: Explain the difference between `range()` and `xrange()`

For the most part, `xrange` and `range` are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use. The only difference is that `range` returns a Python list object and `xrange` returns an `xrange` object.

This means that `xrange` doesn't actually generate a static list at run-time like `range` does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators.

Question 14: Explain the differences between Flask and Django

Django is a Python web framework that offers an open-source, high-level framework that “encourages rapid development and clean, pragmatic design.” It's fast, secure, and scalable. Django offers strong community support and detailed documentation.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) to learn more.

The framework is an inclusive package, in which you get an admin panel, database interfaces, and directory structure right when you create the app. Furthermore, it includes many features, so you don't
Got it!

have to add separate libraries and dependencies. Some features it offers are user authentication, templating engine, routing, database schema migration, and much more.

The Django framework is incredibly flexible in which you can work with MVPs to larger companies. For some perspective, some of the largest companies that use Django are Instagram, Dropbox, Pinterest, and Spotify.

Flask is considered a microframework, which is a minimalistic web framework. It's less "batteries-included," meaning that it lacks a lot of features and functionality that full-stack frameworks like Django offer, such as a web template engine, account authorization, and authentication.

Flask is minimalistic and lightweight, meaning that you add extensions and libraries that you need as you code without automatically being provided with it by the framework. The philosophy behind Flask is that it gives only the components you need to build an app so that you have the flexibility and control. In other words, it's un-opinionated. Some features it offers are a built-in dev server, Restful request dispatching, Http request handling, and much more.

Question 15: What is PYTHONPATH?

It is an environment variable, which is used when a module is imported. Whenever a module is imported, PYTHONPATH is also looked up to check for the presence of the imported modules in various directories. The interpreter uses it to determine which module to load.

Question 16: What is PEP 8?

PEP stands for Python Enhancement Proposal. It is a set of rules that specify how to format Python code for maximum readability.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

Question 17: What are Python decorators?

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.

Question 18: What is init?

`__init__` is a method or constructor in Python. This method is automatically called to allocate memory when a new object/instance of a class is created. All classes have the `__init__` method.

Question 19: What is the ternary operator?

The ternary operator is a way of writing conditional statements in Python. As the name ternary suggests, this Python operator consists of three operands.

Note: The ternary operator can be thought of as a simplified, one-line version of the if-else statement to test a condition.

Syntax

The three operands in a ternary operator include:

- **condition:** A boolean expression that evaluates to either true or false.
- **true_val :** A value to be assigned if the expression is evaluated to true.
- **false_val :** A value to be assigned if the expression is evaluated to false.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

```
var = true_val if condition else false_val
```

Got it!

The variable `var` on the left-hand side of the `=` (assignment) operator will be assigned:

- `value1` if the `booleanExpression` evaluates to `true`.
- `value2` if the `booleanExpression` evaluates to `false`.

Example

```
1 # USING TERNARY OPERATOR
2 to_check = 6
3 msg = "Even" if to_check%2 == 0 else "Odd"
4 print(msg)
5
6 # USING USUAL IF-ELSE
7 msg = ""
8 if(to_check%2 == 0):
9     msg = "Even"
10 else:
11     msg = "Odd"
12 print(msg)
```



Explanation

The above code is using the ternary operator to find if a number is even or odd.

- `msg` will be assigned “even” if the condition `(to_check % 2 == 0)` is `true`.
- `msg` will be assigned “odd” if the condition `(to_check % 2 == 0)` is `false`.

Question 20: What are global and local variables in Python?

Global Variables:

Variables declared outside a function or in global space are called global variables. These variables can be accessed by any function in the program.

Local Variables:

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

Got it!

Any variable declared inside a function is known as a local variable. This variable is present in the local space and not in the global space.

Question 21: What is the `@property` in Python?

The `@property` is a decorator. In Python, decorators enable users to use the class in the same way (irrespective of the changes made to its attributes or methods). The `@property` decorator allows a function to be accessed like an attribute.

Question 22: How is `try/except` used in Python?

An exception is an error that occurs while the program is executing. When this error occurs, the program will stop and generate an exception which then gets handled in order to prevent the program from crashing.

The exceptions generated by a program are caught in the `try` block and handled in the `except` block.

- `Try` : Lets you test a block of code for errors.
- `Except` : Lets you handle the error.

Question 23: Explain the differences between Python 2 and Python 3

Python 2	Python 3
----------	----------

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

Python 2	Python 3
<p>String Encoding</p> <p>Python 2 stores them as ASCII. Unicode is a superset of ASCII and hence, can encode more characters including foreign ones.</p>	<p>String Encoding</p> <p>Python 3 stores strings as Unicode by default.</p>
<p>Division</p> <p>Python 2 division applies the floor function to the decimal output and returns an integer. So dividing 5 by 2 would return <code>floor(2.5) = 2</code>.</p>	<p>Division</p> <p>Division in Python 3 returns the expected output, even if it is in decimals.</p>
<p>Printing</p> <p>Python 2 does not require parentheses.</p>	<p>Printing</p> <p>The syntax for the print statement is different in Python 2 and 3. Python 3 requires parentheses around what is to be printed.</p>
<p>Libraries</p> <p>Many older libraries were built specifically for Python 2 and are not “forward compatible.”</p>	<p>Libraries</p> <p>Some newer libraries are built specifically for Python 3 and do not work with Python 2.</p>

Python 2 is entrenched in the software landscape to the point that co-dependency between several softwares makes it almost impossible to make the shift.

Question 24: What is the join method in python?

We use cookies to enhance your browsing experience. Please click [here](#) to learn more.

The join method in Python takes elements of an iterable and joins them into a string by using a particular string connector value.

Got it!

How does join work?

The join method in Python is a string method, which connects elements of a string iterable structure, which also contains strings or characters (array, list, etc.) by using a particular string as the connector.



Example: Connecting elements using an empty string

This will join the elements in an array using an empty string between each element.

```
1 array = ['H', 'E', 'L', 'L', 'O']
2 connector = ""
3 joined_string = connector.join(array)
4 print(joined_string)
```



Keep the learning going.

Practice the most common Python interview questions and watch your confidence soar. Educative's text-based courses are easy to skim and feature live in-browser coding environments - making learning quick and efficient.

Decode the Coding Interview in Python: Real-World Examples
(<https://www.educative.io/courses/decode-coding-interview-python>)

Question 25: What is dictionary comprehension?

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

Dictionary comprehension is one way to create a dictionary in Python. It creates a dictionary by merging two sets of data which are in the form of either lists or arrays.

The data of one of the two lists/arrays will act as the keys of the dictionary while the data of the second list/array will act as the values. Each key acts as a unique identifier for each value, hence the size of both lists/arrays should be the same.

Here we'll look at simple merging:

Simple merging is merging or combining two lists without any restrictions. In other words, it is the unconditional merging.

The general syntax is as follows:

Example

The following example runs for the college's data base and uses simple merging. Imagine that there is a college's database storing lots of data. For example student's address, grades, section, fees, roll number and so on. Now we need to identify each student uniquely and create a new dictionary which stores all students only. Our decision simply depends on two questions:

- What should be the key?
- What should be the value?

Here we will choose roll numbers as key and names as the value because roll numbers are unique and names could be repetitive. So Alex's roll number is 122 so the tuple will look like 122: Alex. This will be better explained once you try the code attached below.

```
1 rollNumbers =[122,233,353,456]
2 names = ['alex','bob','can', 'don']
3 NewDictionary={ i:j for (i,j) in zip (rollNumbers,names)}
4 print(NewDictionary)
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

Got it!

Question 26: How would you make a deep copy in Python?

A deep copy refers to cloning an object. When we use the = operator, we are not cloning the object; instead, we reference our variable to the same object (a.k.a. shallow copy).

This means that changing one variable's value affects the other variable's value because they are referring (or pointing) to the same object. This difference between a shallow and a deep copy is only applicable to objects that contain other objects, like lists and instances of a class.

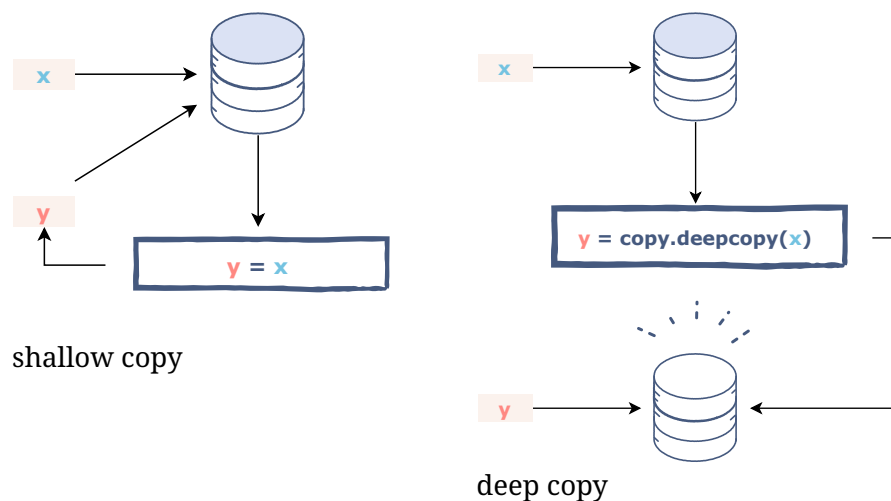
Method

To make a deep copy (or clone) of an object, we import the built-in copy module in Python. This module has the `deepcopy()` method which simplifies our task.

Syntax

This function takes the object we want to clone as its only argument and returns the clone.

Syntax of `copy.deepcopy()`



```
1 import copy
```

```
2 # Using '=' operator
```

```
3 x = [1, 2, 3]
```

```
4 y = x
```

```
5 x[0] = 5 # value of 'y' also changes as it is the SAME c
```

```
6 x[1] = 15
```

Got it!

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.


```
8 print "Shallow copy: ", y
9
10 # Using copy.deepcopy()
11 a = [10, 20, 30]
12 b = copy.deepcopy(a)
13 a[1] = 70 # value of 'b' does NOT change because it is AM
14 print "Deep copy: ", b
```



Question 27: How would you check if a key exists in a Python dictionary?

It is a safe practice to check whether or not the key exists in the dictionary prior to extracting the value of that key. For that purpose, Python offers two built-in functions:

- `has_key()`

The `has_key` method returns true if a given key is available in the dictionary; otherwise, it returns false.

```
1 Fruits = {'a': "Apple", 'b':"Banana", 'c':"Carrot"}
2 key_to_lookup = 'a'
3 if Fruits.has_key(key_to_lookup):
4     print "Key exists"
5 else:
6     print "Key does not exist"
```



- `if-in` statement

This approach uses the `if-in` statement to check whether or not a given key exists in the dictionary.

```
Fruits = {'a': "Apple", 'b':"Banana", 'c':"Carrot"}
key_to_lookup = 'a'
if key_to_lookup in Fruits:
    print "Key exists"
else:
    print "Key does not exist"
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.



Got it!

Question 28: How would you achieve memoization in Python?

Consider this computationally expensive code:

```
# Fibonacci Numbers
def fib(num):
    if num == 0:
        return 0
    elif num == 1:
        return 1
    return fib(num - 1) + fib(n - 2)
```

Memoization can be achieved through Python decorators

Here's the full implementation.

```
1 import timeit
2
3 def memoize_fib(func):
4     cache = {}
5     def inner(arg):
6         if arg not in cache:
7             cache[arg] = func(arg)
8         return cache[arg]
9     return inner
10
11
12 def fib(num):
13     if num == 0:
14         return 0
15     elif num == 1:
16         return 1
17     else:
18         return fib(num-1) + fib(num-2)
19
20 fib = memoize_fib(fib)
21
22
23 print(timeit.timeit('fib(30)', globals=globals(), number=1))
24 print(timeit.timeit('fib(30)', globals=globals(), number=1))
25 print(timeit.timeit('fib(30)', globals=globals(), number=1))
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

Got it!

Question 29: How would you sort a dictionary in Python?

We can sort this type of data by either the key or the value and this is done by using the `sorted()` function.

First, we need to know how to retrieve data from a dictionary to be passed on to this function.

There are two basic ways to get data from a dictionary:

`Dictionary.keys()` : Returns only the keys in an arbitrary order.






`Dictionary.values()` : Returns a list of values. `Dictionary.items()` : Returns all of the data as a list of key-value pairs.

Sorted() syntax This method takes one mandatory and two optional arguments:

Data (mandatory): The data to be sorted. We will pass the data we retrieved using one of the above methods.

Key (optional): A function (or criteria) based on which we would like to sort the list. For example, the criteria could be sorting strings based on their length or any other arbitrary function. This function is applied to every element of the list and the resulting data is sorted. Leaving it empty will sort the list based on its original values.

Reverse (optional): Setting the third parameter as `true` will sort the list in descending order. Leaving this empty sorts in ascending order.

 <code>keys()</code>	 <code>values()</code>	 <code>items()</code>
<pre>dict = {} dict['1'] = 'apple' dict['3'] = 'orange' dict['2'] = 'pango' lst = dict.keys() # Sorted by key print("Sorted by key: ", sorted(lst))</pre>		
<div> </div>		

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!


Question 30: How and when would you use `any()` and `all()`?

What is `any()` ?

`any()` is a function that takes in an iterable (such as a list, tuple, set, etc.) and returns `True` if any of the elements evaluate to `True`, but it returns `False` if all elements evaluate to `False`.

Passing an iterable to `any()` to check if any of the elements are `True` can be done like this:

```
1 one_truth = [True, False, False]
2 three_lies = [0, '', None]
3
4 print(any(one_truth))
5
6 print(any(three_lies))
```



The first print statement prints `True` because one of the elements in `one_truth` is `True`.

On the other hand, the second print statement prints `False` because none of the elements are `True`, i.e., all elements are `False`.

Use `any()` when you need to check a long series of or conditions.


What is `all()` ?

`all()` is another Python function that takes in an iterable and returns `True` if all of the elements evaluate to `True`, but returns `False` if otherwise.

Similar to `any()`, `all()` takes in a list, tuple, set, or any iterable, like so:

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

```
1 all_true = [True, 1, 'a', object()]
2 one_true = [True, False, False, 0]
3 all_false = [None, '', False, 0]
4
5 print(all(all_true))
```



Got it!

```
5 print(all(one_true))  
6 print(all(one_true))  
7 print(all(all_false))
```



The first function call returned `True` because `all_true` was filled with truthy values.

Passing `one_true` to `all()` returned `False` because the list contained one or more falsy values.

Finally, passing `all_false` to `all()` returns `False` because it also contained one or more falsy values.

Use `all()` when you need to check a long series of and conditions.

Question 31: What is a Python Docstring?

The Python docstrings provide a suitable way of associating documentation with:

- Python modules
- Python functions
- Python classes

It is a specified document for the written code. Unlike conventional code comments, the doctoring should describe what a function does, not how it works.

The docstring can be accessed using

- `__doc__` method of the object
- `help` function

Example

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(/privacy\)](#) to learn more.

```
1 def hello_world():  
2     """Demonstrating docstring."""  
3     Got it!  
4     return None
```



```
5
6 # printing using __doc__ method
7 print "Using __doc__ method:"
8 print hello_world.__doc__
9
10 # printing using help function
11 print "Using help function:"
12 help(hello_world)
```



Question 32: Write a Python function and explain what's going on.

Here's our function:

```
def Examplefunc(str): #function that outputs the str parameter
    print "The value is", str
    #no return statement needed in this function

def Multiply(x,y): #function that computes the product of x and y
    return x*y #returning the product of x and y

#Calling the functions
Examplefunc(9) #9 passed as the parameter
answer = Multiply(4,2) #4 and 2 passed as the parameters
print "The product of x and y is:",answer
```



Explanation

The function `Examplefunc` above takes a variable `str` as parameter and then prints this value. Since it only prints the value there is no need for a return command.

The function `Multiply` takes two parameters `x` and `y` as parameters. It then computes the product and uses the `return` statement to return back the answer.

Question 33: Explain the difference between a generator and an iterator in Python.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

Got it!

An iterator in Python serves as a holder for objects so that they can be iterated over; a generator facilitates the creation of a custom iterator.



Apart from the obvious syntactic differences, the following are some noteworthy differences:

Generator	Iterator
Implemented using a function.	Implemented using a class.
Uses the <code>yield</code> keyword.	Does not use the <code>yield</code> keyword.
Usage results in a concise code.	Usage results in a relatively less concise code.
All the local variables before the <code>yield</code> statements are stored.	No local variables are used.

Implementation of Iterator

```

1 # Function to generate all the non-negative numbers
2 # up to the given non-negative number.
3 class UpTo:
4     # giving the parameter a default value of 0
5     def __init__(self, max = 0):
6         self.max = max
7     def __iter__(self):
8         self.n = 0
9         return self
10    def __next__(self):
11        # The next method will throw an
12        # exception when the termination condition is reached
13        if self.n > self.max:
14            raise StopIteration
15        else:
16            result = self.n
17            self.n += 1
18            return result
19    for number in UpTo(5):
20        print(number)

```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.

Got it!



Implementation of Generator

```

1 # Function to generate all the non-negative numbers
2 # up to the given non-negative number
3 def upto(n):
4     for i in range(n+1):
5         # The yield statement is what makes a function
6         # a generator
7         yield i
8 for number in upto(5):
9     print(number)

```



Question 34: What is defaultdict in Python?

The Python dictionary, `dict`, contains words and meanings as well as key-value pairs of any data type. The `defaultdict` is another subdivision of the built-in `dict` class.

How is defaultdict different?

The `defaultdict` is a subdivision of the `dict` class. Its importance lies in the fact that it allows each new key to be given a default value based on the type of dictionary being created.

A `defaultdict` can be created by giving its declaration, an argument that can have three values; list, set or int. According to the specified data type, the dictionary is created and when any key, that does not exist in the `defaultdict` is added or accessed, it is assigned a default value as opposed to giving a `KeyError`.

Example

The first code snippet below shows a simple dictionary and how when a key that does not exist in the `dict` is accessed, it gives an error.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.

```

1 dict_demo = dict()
2 print(dict_demo[3])

```

Got it!





Now let's introduce a `defaultdict` and see what happens.

```
1 from collections import defaultdict
2
3 defaultdict_demo = defaultdict(int)
4 print(defaultdict_demo[3])
```



In this case, we have passed `int` as the datatype to the `defaultdict`. Hence, any key that does not already exist in `defaultdict_demo` will be assigned a value of 0, unless a value is defined for it.

Note: You can also have `set` or `list` as the parameters

Question 35: What are Python modules?

A Python module is a Python file containing a set of functions and variables to be used in an application. The variables can be of any type (arrays, dictionaries, objects, etc.)

Modules can be either:

1. Built in
2. User-defined

Benefits of modules in Python

There are a couple of key benefits of creating and using a module in Python:

- Structured Code
 - Code is logically organized by being grouped into one Python file which makes development easier and less

error-prone.

- Code is easier to understand and use.

- Reusability

Got it!

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

- Functionality defined in a single module can be easily reused by other parts of the application. This eliminates the need to recreate duplicate code.

Python Coding Interview Questions

In this section we'll take a look at common coding interview questions that pertain to lists, linked lists, graphs, trees, multithreading/concurrency and more.

Let's dive in.

Question 36: Reverse a string in Python

Let's reverse the string "Python" using the slicing method.

To reverse a string, we simply create a slice that starts with the length of the string, and ends at index 0.

To reverse a string using slicing, write:

```
stringname[stringlength::-1] # method 1
```

Or write without specifying the length of the string:

```
stringname[::-1] # method2
```

The slice statement means start at string length, end at position 0, move with the step -1 (or one step backward).

```
str="Python" # initial string
stringlength=len(str) # calculate length of the list
slicedString=str[stringlength::-1] # slicing
print (slicedString) # print the reversed string
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.
This is just one method to reverse a string in Python. Other two

notable methods are Loop and Use Join.
Got it!

Question 37: Check if a Python string contains another string

There are a couple ways to check this. For this post, we'll look at the `find` method.

The `find` method checks if the string contains a substring. If it does, the method returns the starting index of a substring within the string; otherwise, it returns -1.

The general syntax is:

```
string.find(substring)
```

```
a_string="Python Programming"
substring1="Programming"
substring2="Language"
print("Check if "+a_string+" contains "+substring1+":")
print(a_string.find(substring1))
print("Check if "+a_string+" contains "+substring2+":")
print(a_string.find(substring2))
```



The other two notable methods for checking if a string contains another string are to use `in` operator or use the `count` method.

Question 38: Implement breadth first search (BFS) in Python

Consider the graph which is implemented in the code below:



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

Got it!

```
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = [] # List to keep track of visited nodes.
queue = []    #Initialize a queue

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
bfs(visited, graph, 'A')
```



Explanation

Lines 3-10: The illustrated graph is represented using an adjacency list. An easy way to do this in Python is to use a dictionary data structure, where each vertex has a stored list of its adjacent nodes.

Line 12: `visited` is a list that is used to keep track of visited nodes.

Line 13: `queue` is a list that is used to keep track of nodes currently in the queue.

Line 29: The arguments of the `bfs` function are the `visited` list, the `graph` in the form of a dictionary, and the starting node `A`.

Lines 15-26: `bfs` follows the algorithm described above:

1. It checks and appends the starting node to the `visited` list and the `queue`.
2. Then, while the queue contains elements, it keeps taking out nodes from the queue, appends the neighbors of that node to the queue if they are unvisited, and marks them as visited.

3. This continues until the queue is empty.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Time Complexity

Got it!

Since all of the nodes and vertices are visited, the time complexity for BFS on a graph is $O(V + E)$; where V is the number of vertices and E is the number of edges.

Question 39: Implement depth first search (DFS) in Python

Consider this graph, implemented in the code below:

```
1 # Using a Python dictionary to act as an adjacency list
2 graph = {
3     'A' : ['B','C'],
4     'B' : ['D', 'E'],
5     'C' : ['F'],
6     'D' : [],
7     'E' : ['F'],
8     'F' : []
9 }
10
11 visited = set() # Set to keep track of visited nodes.
12
13 def dfs(visited, graph, node):
14     if node not in visited:
15         print (node)
16         visited.add(node)
17         for neighbour in graph[node]:
18             dfs(visited, graph, neighbour)
19
20 # Driver Code
21 dfs(visited, graph, 'A')
```

Explanation

Lines 2-9: The illustrated graph is represented using an adjacency list - an easy way to do it in Python is to use a dictionary data structure. Each vertex has a list of its adjacent nodes stored.

Line 11: `visited` is a set that is used to keep track of visited nodes.

Line 21: The `dfs` function is called and is passed the `visited` set, the graph in the form of a dictionary, and `A`, which is the starting node.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) to learn more.

Got it!

Lines 13-18: `dfs` follows the algorithm described above:

1. It first checks if the current node is unvisited - if yes, it is appended in the `visited` set.
2. Then for each neighbor of the current node, the `dfs` function is invoked again.
3. The base case is invoked when all the nodes are visited. The function then returns.

Time Complexity

Since all the nodes and vertices are visited, the average time complexity for DFS on a graph is $O(V + E)$, where V is the number of vertices and E is the number of edges. In case of DFS on a tree, the time complexity is $O(V)$, where V is the number of nodes.

Question 40: Implement wildcards in Python

In Python, you can implement wildcards using the `regex` (regular expressions) library.

The dot `.` character is used in place of the question mark `?` symbol. Hence, to search for all words matching the color pattern, the code would look something like as follows.

```
1 # Regular expression library
2 import re
3
4 # Add or remove the words in this list to vary the results
5 wordlist = ["color", "colour", "work", "working",
6             "fox", "worker", "working"]
7
8 for word in wordlist:
9     # The . symbol is used in place of ? symbol
10    if re.search('col.r', word) :
11        print (word)
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

Got it!

Question 41: Implement merge sort in Python

Here is the code for merge sort in Python:

```

1  def mergeSort(myList):
2      if len(myList) > 1:
3          mid = len(myList) // 2
4          left = myList[:mid]
5          right = myList[mid:]
6
7          # Recursive call on each half
8          mergeSort(left)
9          mergeSort(right)
10
11         # Two iterators for traversing the two halves
12         i = 0
13         j = 0
14
15         # Iterator for the main list
16         k = 0
17
18         while i < len(left) and j < len(right):
19             if left[i] < right[j]:
20                 # The value from the left half has been used
21                 myList[k] = left[i]
22                 # Move the iterator forward
23                 i += 1
24             else:
25                 myList[k] = right[j]
26                 j += 1
27             # Move to the next slot
28             k += 1
29
30         # For all the remaining values
31         while i < len(left):

```

Explanation

This is the recursive approach for implementing merge sort. The steps needed to obtain the sorted array through this method can be found below:

1. The list is divided into `left` and `right` in each recursive call until two adjacent elements are obtained.
2. Now begins the sorting process. The `i` and `j` iterators traverse the two halves in each call. The `k` iterator traverses the whole lists and makes changes along the way.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

3. If the value at `i` is smaller than the value at `j`, `left[i]` is assigned to the `myList[k]` slot and `i` is incremented. If not, then `right[j]` is chosen.

Got it!

4. This way, the values being assigned through `k` are all sorted.
5. At the end of this loop, one of the halves may not have been traversed completely. Its values are simply assigned to the remaining slots in the list.

Time complexity

The algorithm works in $O(n \log n)$. This is because the list is being split in $\log(n)$ calls and the merging process takes linear time in each call.

Question 42: Implement Dijkstra's algorithm in Python

Basic algorithm

1. From each of the unvisited vertices, choose the vertex with the smallest distance and visit it.
2. Update the distance for each neighboring vertex, of the visited vertex, whose current distance is greater than its sum and the weight of the edge between them.
3. Repeat steps 1 and 2 until all the vertices are visited.

For more on Python algorithms for coding interviews, check out our article [Master Algorithms with Python for Coding Interviews \(https://www.educative.io/blog/python-algorithms-coding-interview\)](https://www.educative.io/blog/python-algorithms-coding-interview)

Here's the implementation

```

1 import sys
2
3 # Function to find out which of the unvisited node
4 # needs to be visited next
5 def to_be_visited():
6     global visited_and_distance
7     v = -10
8     # Choosing the vertex with the minimum distance
9     for index in range(number_of_vertices):
10        if visited[index][0] == 0 \
11           and (v < 0 or visited_and_distance[v][1] <= \
12              visited_and_distance[index][1]):
13            v = index
14     return v
15
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.

Got it!


```

16 # Creating the graph as an adjacency matrix
17 vertices = [[0, 1, 1, 0],
18             [0, 0, 1, 0],
19             [0, 0, 0, 1],
20             [0, 0, 0, 0]]
21 edges = [[0, 3, 4, 0],
22          [0, 0, 0.5, 0],
23          [0, 0, 0, 1],
24          [0, 0, 0, 0]]
25
26 number_of_vertices = len(vertices[0])
27
28 # The first element of the lists inside visited_and_distance
29 # denotes if the vertex has been visited.
30 # The second element of the lists inside the visited_and_distance
31 # denotes the distance from the source.

```



Question 43: Merge two sorted lists

```

1 # Merge list1 and list2 and return resulted list
2 def merge_lists(lst1, lst2):
3     index_arr1 = 0
4     index_arr2 = 0
5     index_result = 0
6     result = []
7
8     for i in range(len(lst1)+len(lst2)):
9         result.append(i)
10        # Traverse Both lists and insert smaller value from arr
11        # into result list and then increment that lists index.
12        # If a list is completely traversed, while other one is
13        # copy all the remaining elements into result list
14        while (index_arr1 < len(lst1)) and (index_arr2 < len(lst2)):
15            if (lst1[index_arr1] < lst2[index_arr2]):
16                result[index_result] = lst1[index_arr1]
17                index_result += 1
18                index_arr1 += 1
19            else:
20                result[index_result] = lst2[index_arr2]
21                index_result += 1
22                index_arr2 += 1
23        while (index_arr1 < len(lst1)):
24            result[index_result] = lst1[index_arr1]
25            index_result += 1
26            index_arr1 += 1
27        while (index_arr2 < len(lst2)):
28            result[index_result] = lst2[index_arr2]
29            index_result += 1
30            index_arr2 += 1
31        return result

```



The solution above is a more intuitive way to solve this problem.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

1. Start by creating a new empty list. This list will be filled with all the elements of both lists in sorted order and returned.

Got it!

2. Then initialize three variables to zero to store the current index of each list.
3. Then compare the elements of the two given lists at the current index of each, append the smaller one to the new list and increment the index of that list by 1.
4. Repeat until the end of one of the lists is reached and append the other list to the merged list.

Time Complexity The time complexity for this algorithm is $O(n+m)$ where n and m are the lengths of the lists. This is because both lists are iterated over at least once.

Note that this problem can also be solved by merging in place.

Question 44: Find two numbers that add up to 'k'

```

1  def binarySearch(a, item, curr):
2      first = 0
3      last = len(a) - 1
4      found = False
5      index = -1
6      while first <= last and not found:
7          mid = (first + last) // 2
8          if a[mid] == item:
9              index = mid
10             found = True
11         else:
12             if item < a[mid]:
13                 last = mid - 1
14             else:
15                 first = mid + 1
16     if found:
17         return index
18     else:
19         return -1
20
21
22  def findSum(lst, k):
23      lst.sort()
24      for j in range(len(lst)):
25          # find the difference in list through binary search
26          # return the only if we find an index
27          index = binarySearch(lst, k - lst[j], j)
28          if index is not -1 and index is not j:
29              return [lst[j], k - lst[j]]
30
31

```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.



Got it!

You can solve this problem by first sorting the list. Then for each element in the list, use a binary search to look for the difference between that element and the intended sum. In other words, if the intended sum is k and the first element of the sorted list is a_0 , then we will do a binary search for $k - a_0$. The search is repeated for every a_i up to a_n until one is found. You can implement the `binarySearch()` function however you like, recursively or iteratively.

Time Complexity

Since most optimal comparison-based sorting functions take $O(n \log n)$, let's assume that the Python `.sort()` function takes the same. Moreover, since binary search takes $O(\log n)$ time for finding a single element, therefore a binary search for all n elements will take $O(n \log n)$ time.

Question 45: Find the first non-repeating integer in a list

Here you can use a Python dictionary to keep count of repetitions

Sample input:

```
[9,2,3,2,6,6]
```

```

1 def findFirstUnique(lst):
2     counts = {} # Creating a dictionary
3     # Initializing dictionary with pairs like (lst[i],(count,order))
4     counts = counts.fromkeys(lst, (0,len(lst)))
5     order = 0
6     for ele in lst:
7         # counts[ele][0] += 1 # Incrementing for every repetition
8         # counts[ele][1] = order
9         counts[ele] = (counts[ele][0]+1 , order)
10        order += 1 # increment order
11    answer = None
12    answer_key = None
13    # filter non-repeating with least order
14    for ele in lst:
15        if (counts[ele][0] is 1) and (answer is None):
16            answer = counts[ele]
17            answer_key = ele
18        elif answer is None:
19            continue
20        elif (counts[ele][0] is 1) and (counts[ele][1] < answer[1]):
21            answer = counts[ele]
22            answer_key = ele
23    return answer_key
24
25
```



We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

```
26 print(findFirstUnique([1, 1, 1, 2]))
```



The keys in the `counts` dictionary are the elements of the given list and the values are the number of times each element appears in the list. We return the element that appears at most once in the list on line 23. We need to maintain the order of update for each key in a tuple value.

Time Complexity

Since the list is only iterated over only twice and the `counts` dictionary is initialized with linear time complexity, therefore the time complexity of this solution is linear, i.e., $O(n)$.

Question 46: Find the middle value of the linked list

main.py
LinkedList.py
Node.py

```

1  from Node import Node
2
3
4  class LinkedList:
5      def __init__(self):
6          self.head_node = None
7
8      def get_head(self):
9          return self.head_node
10
11     def is_empty(self):
12         if self.head_node is None:
13             return True
14         else:
15             return False
16
17     def insert_at_head(self, data):
18         temp_node = Node(data)
19         temp_node.next_element = self.head_node
20         self.head_node = temp_node
21         return self.head_node
22
23     def print_list(self):
24         if self.is_empty():
25             print("List is empty")
26             return False
27         temp = self.head_node
28         while temp.next_element is not None:
29             print(temp.data)
30             temp = temp.next_element
31         print(temp.data, '\n')
```

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.

Got it!



Here you can use two pointers which will work simultaneously.

Think of it this way:

- The fast pointer moves two steps at a time till the end of the list
- The slow pointer moves one step at a time
- When the fast pointer reaches the end, the slow pointer will be at the middle

Using this algorithm, you can make the process faster because the calculation of the length and the traversal until the middle are happening side-by-side.

Time Complexity

You are traversing the linked list at twice the speed, so it is certainly faster. However, the bottleneck complexity is still $O(n)$.

Question 47: Reverse first 'k' elements of a queue

main.py

Stack.py

Queue.py

```

1  from Queue import myQueue
2  from Stack import myStack
3
4  # 1.Push first k elements
5  # 2.Pop Stack elements and
6  # 3.Dequeue queue element:
7
8
9  def reverseK(queue, k):
10     if queue.isEmpty() is
11         # Handling invalid
12         return None
13
14     stack = myStack()
15     for i in range(k):
16         stack.push(queue.ele
17
18     while stack.isEmpty()
19         queue.enqueue(stack
20
21     size = queue.size()
22
23     for i in range(size -
24         queue.enqueue(que
25
26     return queue
27
28 Got it!
29 # testing our logic

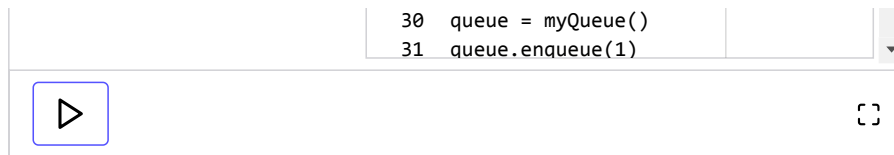
```

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.

```

30 queue = myQueue()
31 queue.enqueue(1)

```



Explanation

1. Check for invalid input, i.e., if the queue is empty, if k is greater than the queue, and if k is negative on **line 2**. If the input is valid, start by creating a Stack. The available stack functions are:
 - **Constructor:** `myStack()`
 - **Push Elements:** `push(int)` to add elements to the stack.
 - **Pop elements:** `pop()` to remove or pop the top element from the stack.
 - **Check if empty:** `isEmpty()` returns true if the stack is empty and false otherwise.
 - **Return back:** `back()` returns the element that has been added at the end without removing it from the stack.
 - **Return front:** `front()` returns the top element (that has been added at the beginning) without removing it from the stack.
2. Our function `reverseK(queue, k)` takes queue as an input parameter. k represents the number of elements we want to reverse. The available queue functions are:
 - **Constructor:** `myQueue(size)` size should be an integer specifying the size of the queue.
 - **Enqueue:** `enqueue(int)`
 - **Dequeue:** `dequeue()`
 - **Check if empty:** `isEmpty()`
 - **Check size:** `size()`
3. Now, moving on to the actual logic, dequeue the first k elements from the front of the queue and push them in the stack we created earlier using `stack.push(queue.dequeue())` in line 8.
4. Once all the k values have been pushed to the stack, start popping them and enqueueing them to the back of the queue sequentially. We will do this using `queue.enqueue(stack.pop())`

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) to learn more.

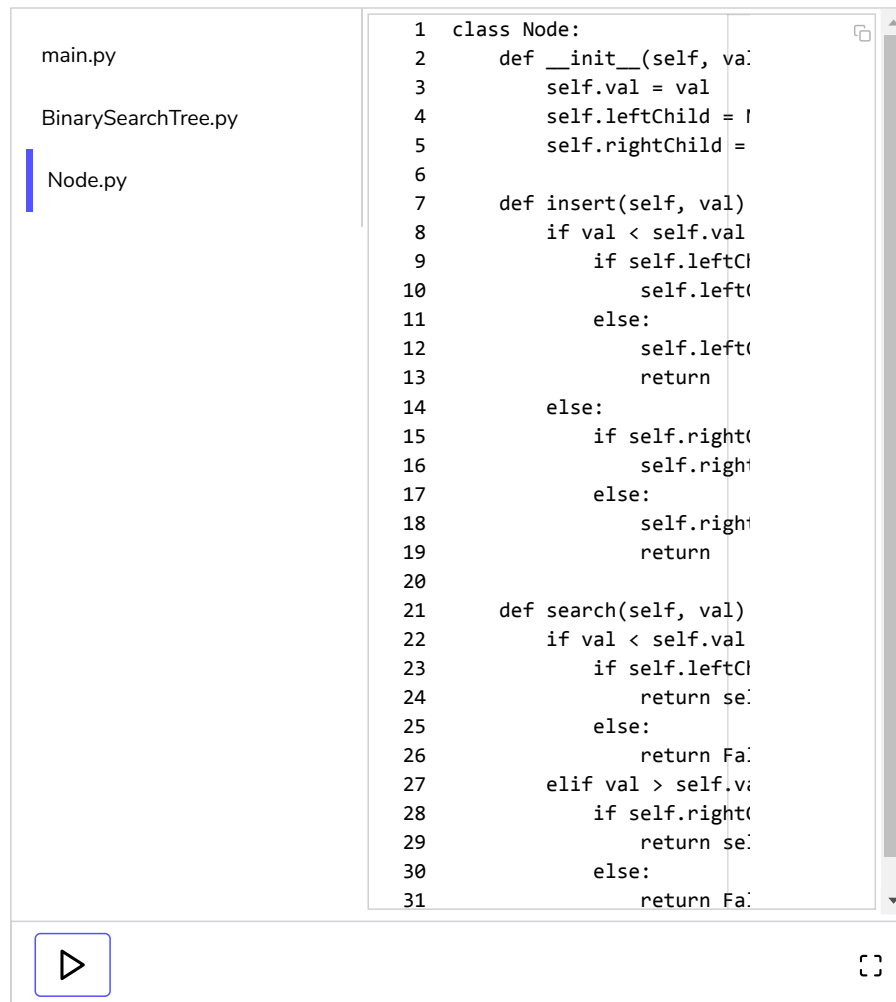
Got it!

in line 12. At the end of this step, we will be left with an empty stack and the k reversed elements will be appended to the back of the queue.

- Now we need to move these reversed elements to the front of the queue. To do this, we used `queue.enqueue(queue.dequeue())` in line 16. Each element is first dequeued from the back

Question 48: Find the height of a binary search tree (BST)

Here you can use recursion to find the heights of the left and right sub-trees.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer lists three files: `main.py`, `BinarySearchTree.py`, and `Node.py`. The `Node.py` file is selected. The code editor displays the following Python code:

```

1 class Node:
2     def __init__(self, val):
3         self.val = val
4         self.leftChild = None
5         self.rightChild = None
6
7     def insert(self, val):
8         if val < self.val:
9             if self.leftChild is None:
10                 self.leftChild = Node(val)
11             else:
12                 self.leftChild.insert(val)
13         else:
14             if self.rightChild is None:
15                 self.rightChild = Node(val)
16             else:
17                 self.rightChild.insert(val)
18         return self
19
20
21     def search(self, val):
22         if val < self.val:
23             if self.leftChild is None:
24                 return False
25             else:
26                 return self.leftChild.search(val)
27         elif val > self.val:
28             if self.rightChild is None:
29                 return False
30             else:
31                 return self.rightChild.search(val)

```

At the bottom of the code editor, there is a play button icon and a full-screen icon.

Explanation

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

Here, we return -1 if the given node is None. Then, we call the `findHeight()` function on the left and right subtrees and return the one that has a greater value plus 1. We will not return 0 if the given node is None as the leaf node will have a height of 0.

Time Complexity

The time complexity of the code is $O(n)O(n)$ as all the nodes of the entire tree have to be traversed.

Question 49: Convert max heap to min heap

Here we'll Min Heapify all Parent Nodes.

```
1 def minHeapify(heap, index):
2     left = index * 2 + 1
3     right = (index * 2) + 2
4     smallest = index
5     # check if left child exists and is less than smallest
6     if len(heap) > left and heap[smallest] > heap[left]:
7         smallest = left
8     # check if right child exists and is less than smallest
9     if len(heap) > right and heap[smallest] > heap[right]:
10        smallest = right
11    # check if current index is not the smallest
12    if smallest != index:
13        # swap current index value with smallest
14        tmp = heap[smallest]
15        heap[smallest] = heap[index]
16        heap[index] = tmp
17        # minHeapify the new node
18        minHeapify(heap, smallest)
19    return heap
20
21
22 def convertMax(maxHeap):
23     # iterate from middle to first element
24     # middle to first indices contain all parent nodes
25     for i in range((len(maxHeap))//2, -1, -1):
26         # call minHeapify on all parent nodes
27         maxHeap = minHeapify(maxHeap, i)
28     return maxHeap
29
30
31 maxHeap = [9, 4, 7, 1, -2, 6, 5]
```



Explanation

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (privacy) to learn more.

Got it!

Remember that we can consider the given `maxHeap` to be a regular list of elements and reorder it so that it represents a min heap accurately. We do exactly that in this solution. The `convertMax()` function restores the heap property on all the nodes from the lowest parent node by calling the `minHeapify()` function on each.

Time Complexity The `minHeapify()` function is called for half of the nodes in the heap. The `minHeapify()` function takes $O(\log(n))$ time and its called on $\frac{n}{2}$ nodes so this solution takes $O(n\log(n))$ time.

Question 50: Detect loop in a linked list

main.py
LinkedList.py
Node.py

```

1  from LinkedList import Li
2  from Node import Node
3
4
5  def detect_loop(lst):
6      # Used to store nodes
7      visited_nodes = set()
8      current_node = lst.get
9
10     # Traverse the set and
11     # and if a node appear
12     # then it means there
13     while current_node:
14         if current_node in
15             return True
16         visited_nodes.add
17         current_node = cur
18     return False
19
20 # -----
21
22
23 lst = LinkedList()
24
25 lst.insert_at_head(21)
26 lst.insert_at_head(14)
27 lst.insert_at_head(7)
28 print(detect_loop(lst))
29
30 head = lst.get_head()
31 node = lst.get head()

```

▶

Explanation

Iterate over the whole linked list and add each visited node to a

`visited_nodes` set. At every node, we check whether it has been visited or not.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

By principle, if a node is revisited a cycle exists!

Time Complexity

We iterate the list once. On average, lookup in a set takes $O(1)$ time. Hence, the average runtime of this algorithm is $O(n)$. However, in the worst case, lookup can increase up to $O(n)$, which would cause the algorithm to work in $O(n^2)$.

Next Steps

Great job with those questions! While interviews can be stressful, practice is the only way to get prepared.

Educative's course, **Decode the Coding Interview in Python: Real-World Examples** (<https://www.educative.io/courses/decode-coding-interview-python>) has helped countless software engineers prepare and land jobs at Microsoft, Amazon, Google, and others. In this course, you'll practice real-world projects picked to prepare you for the industry's toughest interviews.

By the end, you'll have hands-on experience with all the most tested skills and questions.

Happy learning!

Keep reading about Python and interview prep

- Python Concurrency: Making sense of asyncio (<https://www.educative.io/blog/python-concurrency-making-sense-of-asyncio>)
- Stop Using Excel for Data Analytics: Upgrade to Python (<https://www.educative.io/blog/excel-python-data-analytics>)
- Data Structures and Algorithms in Python (<https://www.educative.io/courses/ds-and-algorithms-in-python>)
- CodingInterview.com (<https://www.codinginterview.com/>)

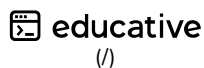
We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) ([privacy](#)) to learn more.

WRITTEN BY

Got it!

Cameron Wilson

Join a community of 500,000 monthly readers. A free, bi-monthly email with a roundup of Educative's top articles and coding tips.



Learn in-demand tech skills in half the time

LEARN

Courses

(/explore)

Early Access Courses

(/explore/early-access)

Edpresso

(/edpresso)

Blog

(/blog)

Pricing

(/unlimited)

For Business

(/business)

CodingInterview.com

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (//codinginterview.com/) to learn more.

For Students
(/github-students)

For Educators
(/github-educators)

CONTRIBUTE

Become an Author
(/authors)

Become an Affiliate
(/affiliate)

LEGAL

Privacy Policy
(/privacy)

Terms of Service
(/terms)

Business Terms of Service
(/enterprise-terms)

MORE

Our Team
(/team)

Careers
(//angel.co/educativeinc/jobs)

For Bootcamps
(/try.educative.io/bootcamps)

Blog for Business
(/blog/enterprise)

Quality Commitment
(/quality)

FAQ

(/courses/educative-faq)

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy](#) (/[privacy](#)) to learn more.

(/contactUs)
Got it!



/educativeinc) (<https://www.linkedin.com/company/educative-inc/>)



(<https://twitter.com/educativeinc>)



(https://www.youtube.com/channel/UCT_8FqzTlr2Q1BOtvX_DPPw/?sub_confirmation=1)



([https://educativesessi](#))

Copyright ©2021 Educative, Inc. All rights reserved.

We use cookies to ensure you get the best experience on our website. Please review our [Privacy Policy \(privacy\)](#) to learn more.

Got it!