Sheena FOLLOW
Python expert with a focus on web technologies, microservices and devops. I also d...

15 Essential Python Interview Questions

Published Nov 19, 2015

Introduction

Looking for a Python job? Chances are you will need to prove that you know how to work with Python. Here are a couple of questions that cover a wide base of skills associated with Python. Focus is placed on the language itself, and not any particular package or framework. Each question will be linked to a suitable tutorial if there is one. Some questions will wrap up multiple topics.

I haven't actually been given an interview test quite as hard as this one, if you can get to the answers comfortably then go get yourself a job.

What This Tutorial Is Not

This tutorial does not aim to cover every available workplace culture - different ons in different ways; they will follow different ings. They will test you in different ways. om of a computer and ask you to solve simple nt of a white board and do similar; some will e will just have a conversation with you.

lly programming. This is a difficult thing to points make sure that you can actually use uestions. If you actually understand how to

Similarly, the best test for a software engineer is actually engineering. This tutorial is about Python as a language. Being able to design efficient, effective, maintainable class hierarchies for solving niche problems is great and wonderful and a skill set worth pursuing but well beyond the scope of this text.

Another thing this tutorial is not is PEP8 compliant. This is intentional as, as mentioned before, different employers will follow different conventions. You will need to adapt to fit the culture of the workplace. Because practicality beats purity.

Another thing this tutorial isn't is concise. I don't want to just throw questions and answers at you and hope something sticks. I want you to get it, or at least get it well enough that you are in a position to look for further explanations yourself for any problem topics.

Question 1

What is Python really? You can (and are encouraged) make comparisons to other technologies in your answer

Answer

Here are a few key points:

- Python is an interpreted language. That means that, unlike languages like *C* and its variants, Python does not need to be compiled before it is run. Other interpreted languages include *PHP* and *Ruby*.
- Python is dynamically typed, this means that you don't need to state the types of anything like that. You can do things like " without error

ated programming in that it allows the osition and inheritance. Python does not have , `private`), the justification for this point is

jects. This means that they can be assigned to ons and passed into functions. Classes are

- Writing Python code is quick but running it is often slower than compiled languages. Fortunately， Python allows the inclusion of C based extensions so bottlenecks can be optimised away and often are. The `numpy` package is a good example of this, it's really quite quick because a lot of the number crunching it does isn't actually done by Python
- Python finds use in many spheres - web applications, automation, scientific modelling, big data applications and many more. It's also often used as "glue" code to get other languages and components to play nice.
- Python makes difficult things easy so programmers can focus on overriding algorithms and structures rather than nitty-gritty low level details.

Why This Matters:

If you are applying for a Python position, you should know what it is and why it is so gosh-darn cool. And why it isn't o.O

Question 2

Fill in the missing code:

```
def print_directory_contents(sPath):
    """
    This function takes the name of a directory
    and prints out the paths files within that
    directory as well as any files contained in
    contained directories.

    This function is similar to os.walk. Please don't
    use os.walk in your answer. We are interested in your
    ability to work with nested structures.
    """
    fill_this_in
```


```
def print_directory_contents(sPath):
    import os
    for sChild in os.listdir(sPath):
        sChildPath = os.path.join(sPath,sChild)
        if os.path.isdir(sChildPath):
            print_directory_contents(sChildPath)
        else:
            print(sChildPath)
```

Pay Special Attention

- Be consistent with your naming conventions. If there is a naming convention evident in any sample code, stick to it. Even if it is not the naming convention you usually use
- Recursive functions need to recurse *and* terminate. Make sure you understand how this happens so that you avoid bottomless callstacks
- We use the `os` module for interacting with the operating system in a way that is cross platform. You could say `sChildPath = sPath + '/' + sChild` but that wouldn't work on windows
- Familiarity with base packages is really worthwhile, but don't break your head trying to memorize everything, Google is your friend in the workplace!
- Ask questions if you don't understand what the code is supposed to do
- KISS! Keep it Simple, Stupid!

Why This Matters:

- Displays knowledge of basic operating system interaction stuff

the final values of A0, A1, ...An.

```
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1 = range(10)
A2 = sorted([i for i in A1 if i in A0])
A3 = sorted([A0[s] for s in A0])
A4 = [i for i in A1 if i in A3]
A5 = {i:i*i for i in A1}
A6 = [[i,i*i] for i in A1]
```

If you dont know what `zip` is don't stress out. No sane employer will expect you to memorize the standard library. Here is the output of `help(zip)` .

```
zip(...)
    zip(seq1 [, seq2 [...]]) -> [(seq1[0], seq2[0] ...), (...)]

    Return a list of tuples, where each tuple contains the i-th elemen
    from each of the argument sequences.  The returned list is truncat
    in length to the length of the shortest argument sequence.
```

If that doesn't make sense then take a few minutes to figure it out however you choose to.

Answer

```
A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}  # the order may vary
A1 = range(0, 10) # or [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] in python 2
A2 = []
A3 = [1, 2, 3, 4, 5]
                                                  16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81
                                                  , 9], [4, 16], [5, 25], [6, 36], [7, 4
```


me saver and a big stumbling block for a lot of

2. If you can read them, you can probably write them down
3. Some of this code was made to be deliberately weird. You may need to work with some weird people

Question 4

Python and multi-threading. Is it a good idea? List some ways to get some Python code to run in a parallel way.

Answer

Python doesn't allow multi-threading in the truest sense of the word. It 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. 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. 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. 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.

There are reasons to use Python's threading package. If you want to run some things simultaneously, and efficiency is not a concern, then it's totally fine and convenient. Or if you are running code that needs to wait for something (like some IO) then it could make a lot of sense. But the threading library won't let you use extra CPU cores.

Multi-threading can be outsourced to the operating system (by doing multi-processing), some external application that calls your Python code (eg, Spark or Hadoop), or some code that your Python code calls (eg: you could have your Python code call a C function that does the expensive multi-threaded stuff).

ople spend a lot of time trying to find
threaded code before they learn what the GIL

ions of your code?

Version control! At this point, you should act excited and tell them how you even use Git (or whatever is your favorite) to keep track of correspondence with Granny. Git is my preferred version control system, but there are others, for example subversion.

Why This Matters:

Because code without version control is like coffee without a cup. Sometimes we need to write once-off throw away scripts and that's ok, but if you are dealing with any significant amount of code, a version control system will be a benefit. Version Control helps with keeping track of who made what change to the code base; finding out when bugs were introduced to the code; keeping track of versions and releases of your software; distributing the source code amongst team members; deployment and certain automations. It allows you to roll your code back to before you broke it which is great on its own. Lots of stuff. It's just great.

Question 6

What does this code output:

```
def f(x,l=[]):
    for i in range(x):
        l.append(i*i)
    print(l)

f(2)
f(3,[3,2,1])
f(3)
```

Answer

obvious, the loop appends 0 and then 1 to the

refers to this new list. It then appends 0, 1 and 4 to this new list. So that's great. The third function call is the weird one. It uses the original list stored in the original memory block. That is why it starts off with 0 and 1.

Try this out if you don't understand:

```
l_mem = []

l = l_mem           # the first call
for i in range(2):
    l.append(i*i)

print(l)            # [0, 1]

l = [3,2,1]         # the second call
for i in range(3):
    l.append(i*i)

print(l)            # [3, 2, 1, 0, 1, 4]

l = l_mem           # the third call
for i in range(3):
    l.append(i*i)

print(l)            # [0, 1, 0, 1, 4]
```

Question 7

What is monkey patching and is it ever a good idea?

Answer

iour of a function or object after it has already

```
datetime.datetime(2012, 12, 12)
```

very useful to this end.

Why This Matters

It shows that you understand a bit about methodologies in unit testing. Your mention of monkey avoidance will show that you aren't one of those coders who favor fancy code over maintainable code (they are out there, and they suck to work with). Remember the principle of KISS? And it shows that you know a little bit about how Python works on a lower level, how functions are actually stored and called and suchlike.

PS: it's really worth reading a little bit about mock if you haven't yet. It's pretty useful.

Question 8

What does this stuff mean: `*args` , `**kwargs` ? And why would we use it?

Answer

Use `*args` when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. `**kwargs` is used when we dont know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers `args` and `kwargs` are a convention, you could also use `*bob` and `**billy` but that would not be wise.

Here is a little illustration:

```
def f(*args,**kwargs): print(args, kwargs)

l = [1,2,3]
t = (4,5,6)
d = {'a':7,'b':8,'c':9}

f()
f(1,2,3)                    # (1, 2, 3) {}
f(1,2,3,"groovy")           # (1, 2, 3, 'groovy') {}
f(a=1,b=2,c=3)              # () {'a': 1, 'c': 3, 'b': 2}
f(a=1,b=2,c=3,zzz="hi")     # () {'a': 1, 'c': 3, 'b': 2, 'zzz': 'hi'}
f(1,2,3,a=1,b=2,c=3)        # (1, 2, 3) {'a': 1, 'c': 3, 'b': 2}

f(*l,**d)                   # (1, 2, 3) {'a': 7, 'c': 9, 'b': 8}
f(*t,**d)                   # (4, 5, 6) {'a': 7, 'c': 9, 'b': 8}
f(1,2,*t)                   # (1, 2, 4, 5, 6) {}
f(q="winning",**d)          # () {'a': 7, 'q': 'winning', 'c': 9, 'b':
f(1,2,*t,q="winning",**d)   # (1, 2, 4, 5, 6) {'a': 7, 'q': 'winning',

def f2(arg1,arg2,*args,**kwargs): print(arg1,arg2, args, kwargs)

f2(1,2,3)                       # 1 2 (3,) {}
f2(1,2,3,"groovy")              # 1 2 (3, 'groovy') {}
f2(arg1=1,arg2=2,c=3)           # 1 2 () {'c': 3}
f2(arg1=1,arg2=2,c=3,zzz="hi")  # 1 2 () {'c': 3, 'zzz': 'hi'}
f2(1,2,3,a=1,b=2,c=3)           # 1 2 (3,) {'a': 1, 'c': 3, 'b': 2}

f2(*l,**d)                    # 1 2 (3,) {'a': 7, 'c': 9, 'b': 8}
f2(*t,**d)                    # 4 5 (6,) {'a': 7, 'c': 9, 'b': 8}
f2(1,2,*t)                    # 1 2 (4, 5, 6) {}
f2(1,1,q="winning",**d)       # 1 1 () {'a': 7, 'q': 'winning', 'c': 9,
f2(1,2,*t,q="winning",**d)    # 1 2 (4, 5, 6) {'a': 7, 'q': 'winning',
```

nown number of arguments or keyword
we will want to store arguments or keyword
just a time saver.

Answer Background Knowledge

These are decorators. A decorator is a special kind of function that either takes a function and returns a function, or takes a class and returns a class. The `@` symbol is just syntactic sugar that allows you to decorate something in a way that's easy to read.

```
@my_decorator
def my_func(stuff):
    do_things
```

Is equivalent to

```
def my_func(stuff):
    do_things

my_func = my_decorator(my_func)
```

You can find a tutorial on how decorators in general work here.

Actual Answer

The decorators `@classmethod`, `@staticmethod` and `@property` are used on functions defined within classes. Here is how they behave:

```
class MyClass(object):
    def __init__(self):
roperties are nice"
y = "VERY nice"
args):
thod({0},{1})".format(args,kwargs))

rgs):
hod({0},{1})".format(args,kwargs))

args):
thod({0},{1})".format(args,kwargs))
```


```
        return self._some_property
    @some_property.setter
    def some_property(self,*args,**kwargs):
        print("calling some_property setter({0},{1},{2})".format(self,
        self._some_property = args[0]
    @property
    def some_other_property(self,*args,**kwargs):
        print("calling some_other_property getter({0},{1},{2})".format
        return self._some_other_property

o = MyClass()
# undecorated methods work like normal, they get the current instance

o.normal_method
# <bound method MyClass.normal_method of <__main__.MyClass instance at

o.normal_method()
# normal_method((<__main__.MyClass instance at 0x7fdd2537ea28>,),{})

o.normal_method(1,2,x=3,y=4)
# normal_method((<__main__.MyClass instance at 0x7fdd2537ea28>, 1, 2),

# class methods always get the class as the first argument

o.class_method
# <bound method classobj.class_method of <class __main__.MyClass at 0x

o.class_method()
# class_method((<class __main__.MyClass at 0x7fdd2536a390>,),{})

o.class_method(1,2,x=3,y=4)
# class_method((<class __main__.MyClass at 0x7fdd2536a390>, 1, 2),{'y'

# static methods have no arguments except the ones you pass in when yo
```

fdd25375848>

'x': 3})

enting getters and setters. It's an er

```
o.some_property
# calling some_property getter(<__main__.MyClass instance at 0x7fb2b70
# 'properties are nice'

o.some_property()
# calling some_property getter(<__main__.MyClass instance at 0x7fb2b70
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# TypeError: 'str' object is not callable

o.some_other_property
# calling some_other_property getter(<__main__.MyClass instance at 0x7
# 'VERY nice'

# o.some_other_property()
# calling some_other_property getter(<__main__.MyClass instance at 0x7
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# TypeError: 'str' object is not callable

o.some_property = "groovy"
# calling some_property setter(<__main__.MyClass object at 0x7fb2b7077

o.some_property
# calling some_property getter(<__main__.MyClass object at 0x7fb2b7077
# 'groovy'

o.some_other_property = "very groovy"
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# AttributeError: can't set attribute

o.some_other_property
# calling some_other_property getter(<__main__.MyClass object at 0x7fb
# 'VERY nice'
```


output?

Enjoy this post?

222

```
        print("stop A stop!")
    def pause(self):
        raise Exception("Not Implemented")

class B(A):
    def go(self):
        super(B, self).go()
        print("go B go!")

class C(A):
    def go(self):
        super(C, self).go()
        print("go C go!")
    def stop(self):
        super(C, self).stop()
        print("stop C stop!")

class D(B,C):
    def go(self):
        super(D, self).go()
        print("go D go!")
    def stop(self):
        super(D, self).stop()
        print("stop D stop!")
    def pause(self):
        print("wait D wait!")

class E(B,C): pass

a = A()
b = B()
c = C()
d = D()
e = E()
```

ds

```
e.stop()

a.pause()
b.pause()
c.pause()
d.pause()
e.pause()
```

Answer

The output is specified in the comments in the segment below:

```
a.go()
# go A go!

b.go()
# go A go!
# go B go!

c.go()
# go A go!
# go C go!

d.go()
# go A go!
# go C go!
# go B go!
# go D go!

e.go()
# go A go!
# go C go!
```


```
d.stop()
# stop A stop!
# stop C stop!
# stop D stop!

e.stop()
# stop A stop!

a.pause()
# ... Exception: Not Implemented

b.pause()
# ... Exception: Not Implemented

c.pause()
# ... Exception: Not Implemented

d.pause()
# wait D wait!

e.pause()
# ...Exception: Not Implemented
```

Why do we care?

Because OO programming is really, really important. Really. Answering this question shows your understanding of inheritance and the use of Python's `super` function. Most of the time the order of resolution doesn't matter. Sometimes it does, it depends on your application.

Question 11

Consider the following code, what will it output?


```
mat(self.sName)
gs):
```

```
        print(self)
        for oChild in self._lChildren:
            oChild.print_all_1()
    def print_all_2(self):
        def gen(o):
            lAll = [o,]
            while lAll:
                oNext = lAll.pop(0)
                lAll.extend(oNext._lChildren)
                yield oNext
        for oNode in gen(self):
            print(oNode)

oRoot = Node("root")
oChild1 = Node("child1")
oChild2 = Node("child2")
oChild3 = Node("child3")
oChild4 = Node("child4")
oChild5 = Node("child5")
oChild6 = Node("child6")
oChild7 = Node("child7")
oChild8 = Node("child8")
oChild9 = Node("child9")
oChild10 = Node("child10")

oRoot.append(oChild1)
oRoot.append(oChild2)
oRoot.append(oChild3)
oChild1.append(oChild4)
oChild1.append(oChild5)
oChild2.append(oChild6)
oChild4.append(oChild7)
oChild3.append(oChild8)
oChild3.append(oChild9)
oChild6.append(oChild10)
```

ds

```
<Node 'root'>
<Node 'child1'>
<Node 'child4'>
<Node 'child7'>
<Node 'child5'>
<Node 'child2'>
<Node 'child6'>
<Node 'child10'>
<Node 'child3'>
<Node 'child8'>
<Node 'child9'>
```

`oRoot.print_all_2()` prints:

```
<Node 'root'>
<Node 'child1'>
<Node 'child2'>
<Node 'child3'>
<Node 'child4'>
<Node 'child5'>
<Node 'child6'>
<Node 'child8'>
<Node 'child9'>
<Node 'child7'>
<Node 'child10'>
```

Why do we care?

Because composition and object construction is what objects are all about. Objects be initialised somehow. This also ties up generators.

chieved similar functionality to `print_all_2` hen printing it's contents. One of the nice n't need to take up much space in memory.

`_all_1` traverses the tree in a depth-first irst. Make sure you understand those terms

Question 12

Describe Python's garbage collection mechanism in brief.

Answer

A lot can be said here. There are a few main points that you should mention:

- Python maintains a count of the number of references to each object in memory. If a reference count goes to zero then the associated object is no longer live and the memory allocated to that object can be freed up for something else
- occasionally things called "reference cycles" happen. The garbage collector periodically looks for these and cleans them up. An example would be if you have two objects `o1` and `o2` such that `o1.x == o2` and `o2.x == o1`. If `o1` and `o2` are not referenced by anything else then they shouldn't be live. But each of them has a reference count of 1.
- Certain heuristics are used to speed up garbage collection. For example, recently created objects are more likely to be dead. As objects are created, the garbage collector assigns them to generations. Each object gets one generation, and younger generations are dealt with first.

This explanation is CPython specific.

Question 13

Place the following functions below in order of their efficiency. They all take in a list of numbers between 0 and 1. The list can be quite long. An example input list would be `[random.random() for i in range(100000)]`. How would you prove that your answer is correct?

```
def f1(lIn):
    l1 = sorted(lIn)
    l2 = [i for i in l1 if i<0.5]
    return [i*i for i in l2]

def f2(lIn):
    l1 = [i for i in lIn if i<0.5]
    l2 = sorted(l1)
    return [i*i for i in l2]

def f3(lIn):
    l1 = [i*i for i in lIn]
    l2 = sorted(l1)
    return [i for i in l1 if i<(0.5*0.5)]
```

Answer

Most to least efficient: `f2`, `f1`, `f3`. To prove that this is the case, you would want to profile your code. Python has a lovely profiling package that should do the trick.

```
import cProfile
lIn = [random.random() for i in range(100000)]
cProfile.run('f1(lIn)')
cProfile.run('f2(lIn)')
cProfile.run('f3(lIn)')
```

For completion's sake, here is what the above profile outputs:

```
>>> cProfile.run('f1(lIn)')
         4 function calls in 0.045 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function
        1    0.009    0.009    0.044    0.044 <stdin>:1(f1)
        1    0.001    0.001    0.045    0.045 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_l
        1    0.035    0.035    0.035    0.035 {sorted}


>>> cProfile.run('f2(lIn)')
         4 function calls in 0.024 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function
        1    0.008    0.008    0.023    0.023 <stdin>:1(f2)
        1    0.001    0.001    0.024    0.024 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_l
        1    0.016    0.016    0.016    0.016 {sorted}


>>> cProfile.run('f3(lIn)')
         4 function calls in 0.055 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function
        1    0.016    0.016    0.054    0.054 <stdin>:1(f3)
        1    0.001    0.001    0.055    0.055 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_l
        1    0.038    0.038    0.038    0.038 {sorted}
```


…en pretty worthwhile. A lot of coding for …se - in the example above it's obviously …so if you have the choice of filtering before a …ous stuff can still be located through use of

Question 14

Something you failed at?

Wrong answer

I never fail!

Why This Is Important:

Shows that you are capable of admitting errors, taking responsibility for your mistakes, and learning from your mistakes. All of these things are pretty darn important if you are going to be useful. If you are actually perfect then too bad, you might need to get creative here.

Question 15

Do you have any personal projects?

Really?

This shows that you are willing to do more than the bare minimum in terms of keeping your skillset up to date. If you work on personal projects and code outside of the workplace then employers are more likely to see you as an asset that will grow. Even if they don't ask this question I find it's useful to broach the subject.

Conclusion

These questions intentionally touched on many topics. And the answers were intentionally verbose. In a programming interview, you will need to demonstrate your understanding and if you can do it in a concise way then by all means do that. I tried to give enough information in the answers that you could glean some meaning from them even if you had never heard of some of these topics before. I hope you find this useful in your job hunt.

Enjoy this post? Give **Sheena** a like if it's helpful.

222 70 SHARE

Sheena

Python expert with a focus on web technologies, microservices and devops. I also do some frontend work (React and Angular experience)

I'm about solving problems. Usually I do that by writing code. Often I do that by leading the efforts of others. I get a lot of satisfaction from the constant learning and puzzle solving that comes with my profession. I get even m...

FOLLOW

70 Replies

Leave a reply

menaka barathi 6 days ago

Good Post and I like the way you explained things, Sheena.

https://www.tibacademy.in/machine-learning-training-in-bangalore/
https://www.tibacademy.in/aws-training-in-bangalore/
https://www.tibacademy.in/hadoop-training-in-bangalore/

Enjoy this post? 222

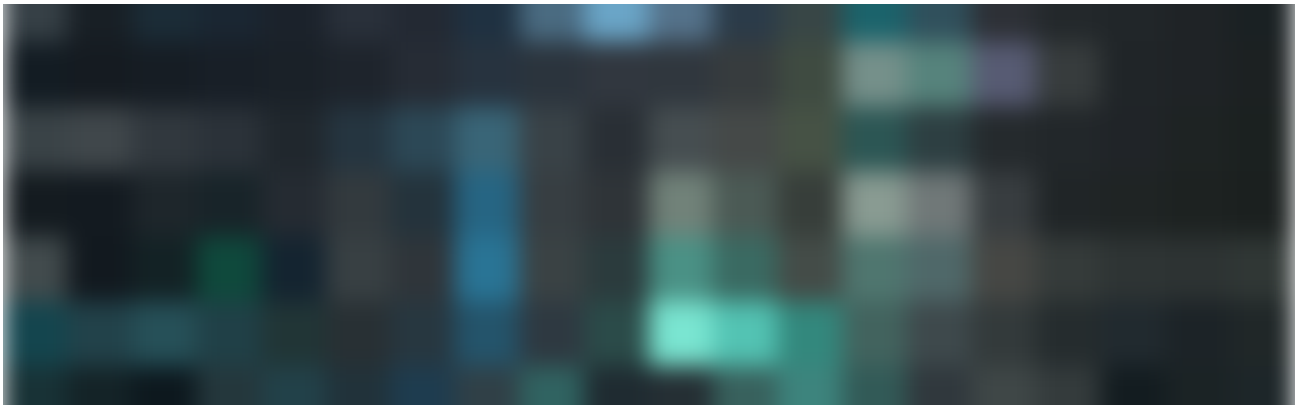
What are mobile marketing and its uses?
WhatsApp is the best marketing tool. All users use in WhatsApp. we always share your points. If you want market-oriented more details to contact us: https://webdesigntraining.co.in/

Reply

Show more replies

Kartik Singh

7 Technical Concept Every Data Science Beginner Should Know



So you want to learn data science but you don't know where to start? **Or you are a beginner and you want to learn the basic concepts? Welcome to your new career and your new life!** You will discover a lot of things on your journey to becoming a data scientist and being part of a new revolution. I am a firm believer that you can

Enjoy this post?

222