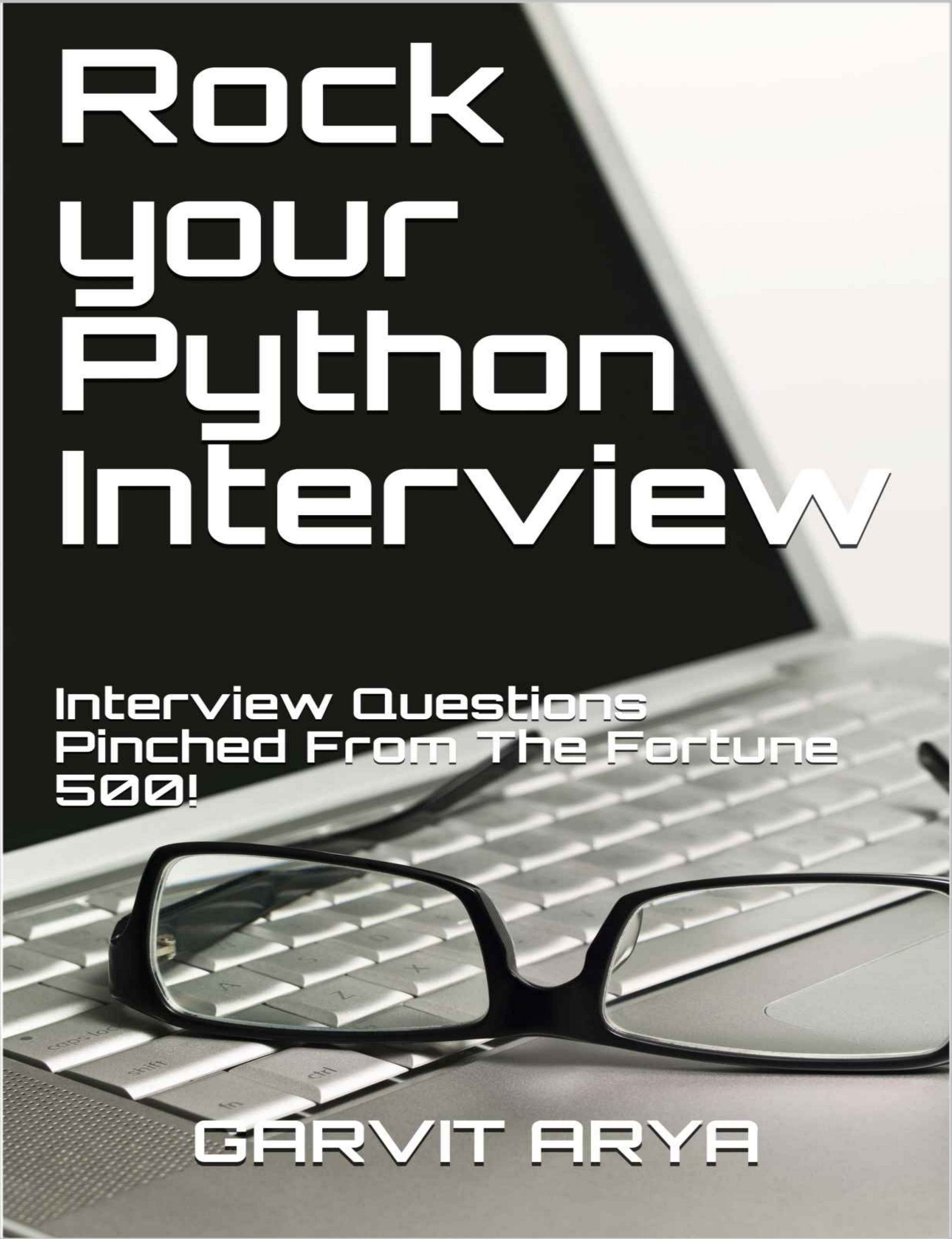

Rock your Python Interview

Interview Questions Pinched From The Fortune 500!

GARVIT ARYA

Rock your Python Interview

Interview Questions Pinched From The Fortune 500!

By Garvit Arya

Epigraph

Looking for a job in Data Science or Analytical role?
Chances are you will need to prove that you know how to work with Python.

As a competent Analyst, or as I prefer to say it as Data Sherpa, a thorough understanding of Python is broadly required and you must be able to answer all Python interview questions with authority and lucidity. To succeed in such Analytical interviews, not only do you have to master the art of undergoing interviews, but also the art of giving all answers with ease.

I hope this set of Python Interview Questions will help you in preparing for your interviews.

If you found this book helpful, then please leave feedback and rate it 5 Stars. If you are planning to review the book, then you can send a screenshot of the review to garvitarya1994@gmail.com and I will send you a FREE copy of my upcoming interview guidebook :)

All the best!

About the Author

I am a Data Sherpa who converts data into insights at day and spends my nights exploring & learning new technologies. I am continuing my self-education with deep-learning courses, enjoy coding for data analysis and visualization projects, and love writing on the topics of data science and artificial intelligence.

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 more satisfaction from contributing to a successful and worthwhile product.

I also get a lot of joy out of mentoring - I've taken people who know nothing about data science to be capable analysts, and I've managed to inspire a few people who thought coding was totally beyond them to take the leap. I'm good at explaining stuff and I like it.

Happy Coding ☺

Table of Contents

About This Book

Python is a very popular language for Data Science and Machine learning projects. A lot of companies are looking for analysts proficient in Python language.

This book contains basic to intermediate level Python interview questions that an interviewer asks. The intention of this book is to provide answers to difficult questions in the most layman-ish term. If you want to know the in-depth definition then you can Google it.

I have compiled this list after helping dozens of my friends to prepare for technical interviews in top-notch companies like - Facebook, Netflix, Apple, Uber, Amazon, Flipkart, Walmart, HSBC, etc.

Some other questions are scraped from different publicly available internet sources like blogs, Reddit posts, GitHub Repos and competitive sites. The idea is to provide a one-stop-shop for all important topics saving hours of redundant effort searching this on the internet from a heap of non-useful stuff.

Often, these questions and concepts are also used in our daily programming work. But these are most helpful when an Interviewer is trying to test your fundamental knowledge of Python.

How will this book help me?
By reading this book, you do not have to spend time searching the Internet for Python interview questions. I have already compiled a list of the most popular and the latest Python Interview questions.

Are there answers in this book?
Yes, in this book each question is followed by an answer. So you can save time in interview preparation.

What is the best way of reading this book?

You have to first do a slow reading of all the questions in this book. Once you go through them in the first pass, mark the questions that you could not answer by yourself. Then, in the second pass, go through only the difficult questions. After going through this book 2-3 times, you will be well prepared to face a technical interview for Data/Business Analyst position in Python-specific role.

What is the level of questions in this book?

This book has both theoretical as well as coding based questions. This book contains questions that are good for a Technical Business Analyst or Data Analyst profile and can be used for T/PgM roles as well. The difficulty level of question varies in the book from Fresher to a Seasoned professional.

Theoretical Interview Questions

Q.1) What is Python? Can you tell us some benefits of using it?
A) Python is a programming language that is primarily used for web development. It is one of the most sought-after programming languages by developers all over the world. Here are some of the inherent benefits of Python:
1. It can interact with a lot of platforms as well as languages.
2. It is free to use, inclusive of uses for commercial purposes.
3. It is extremely easy to learn. The constant support from users of Python from all over the world makes it really simple to help resolve queries if any.
4. Its object-oriented design greatly contributes to the productivity and speed of the programming language.

Q.2) Please point out a few differences between Python 2.x and 3.x?
A) There are more changes/updates in Python 3 than can be mentioned, but here are the most commonly known ones:
1. Change in the printing syntax.
2. Change in the way integer division behaves. E.g. 3/2 = 1 in Python 2 as opposed to 1.5 in Python 3.
3. Introduction of Unicode and two-byte classes.
4. A new method "__contains__" for range object, which helps fasten the lookup process.
5. Enforcement of the new syntax to raise exceptions; i.e. you will get an error if you don't surround the exception argument with parentheses.
6. Deprecation of the .next() method. Only the next function remains.
7. A TypeError is now rightly raised whenever two unorderable types are compared.

Q.3) What is the difference between .py and .pyc files?
A) Python files are Python source files. .pyc files are the compiled bytecode files that is generated by the Python compiler.

Q.4) Does Python Supports for Web Programming?
A) Yes with Django, Pyramid, Bottle, Tornado, Flask, web2py framework

Q.5) Explain the use of try: except raise, and finally.
A) Python makes use of try, except and finally blocks for error handling. Try block is used to execute the code until an error occurs. We can make use of an except block to receive control which will receive control after all errors, or one can use specific exception handling blocks for various error types. Control is transferred to the appropriate except block. In all cases, the final block is executed. Raise may be used to raise your own exceptions.

Q.6) What are the alternatives of FOR loop for simple repetitive computations.
A) Python supports a couple of looping constructs. The FOR statement is most commonly used. It loops over the elements of a sequence, assigning each to the loop variable. If the body of your loop is simple, the interpreter overhead of the for loop itself can be a substantial amount of the overhead. This is where the map function is handy. You can think of map as a for moved into C code. The only restriction is that the "loop body" of map must be a function call. Besides the syntactic benefit of list comprehensions, they are often as fast or faster than equivalent use of map.
Here's a straightforward example. Instead of looping over a list of words and converting them to upper case:
```
newlist = []
for word in oldlist:
    newlist.append(word.upper())
```
you can use map to push the loop from the interpreter into compiled C code:
```
newlist = map(str.upper, oldlist)
```
List comprehensions were added to Python in version 2.0 as well. They provide a syntactically more compact and more efficient way of writing the above for loop:

```
newlist = [s.upper() for s in oldlist]
```

Q.7) What is a Tuple? What is the difference between lists and tuples in Python?
A) Both lists and tuples in Python are sequence data types that can store a collection of items. Each item in a program can be stored in a list or a tuple and can be of any data type. But the major difference between the two of them is that lists are mutable whereas tuples are immutable. This means that in lists, one can modify the values but this cannot be done in a tuple which also means that they cannot be copied. Another major difference is that. tuples are used to store heterogeneous elements, which are elements belonging to different data types. Whereas lists are used to store homogeneous elements, which are elements that belong to the same type. An immutable object can also be used as a key in a dictionary and so, tuples can be used as dictionary keys if needed.

Q.8) What is monkey patching and is it ever a good idea?
A) Monkey patching is changing the behaviour of a function or object after it has already been defined.
For example:
```
    import datetime
        datetime.datetime.now = lambda: datetime.datetime(2012, 12,
12)
```
Most of the time it's a pretty terrible idea - it is usually best if things act in a well-defined way. One reason to monkey patch would be in testing. The mock package is very useful to this end.

Q.9) What is pickling and unpickling?
A) Pickling and unpickling is used for serializing and de-serializing an object structure in Python. Any object in Python can be pickled so that it can be saved on disk. Pickling basically serializes the object into binary streams first, before writing it to file. It is a way to convert a python object into a character stream. This is useful when one wants to save the state of his objects and reuse them at another time without losing any instance-specific data. By a similar logic, when

one restores a saved game, he will be loading data from its pickled state, therefore unpickling it. The real-world uses of Pickling and Unpickling are widespread as they allow you to easily send data from one server to another and eventually store it in a file or database.
Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using the dump function, this process is called pickling. While the process of retrieving original Python objects from the stored string representation is called unpickling.

Q.10) What is the lambda operator?
A) Lambda operator or lambda function is used for creating small, one-time and anonymous function objects in Python. It can have any number of arguments, but it can have only one expression. It cannot contain any statements and it returns a function object which can be assigned to any variable. Mostly lambda functions are passed as parameters to a function which expects a function object as parameter like map, reduce, filter functions

Q.11) What is the difference between range & xrange?
A) 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, however you please. The only difference is that range returns a Python list object and x range 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. That means that if you have a really gigantic range you’d like to generate a list for, say one billion, xrange is the function to use.
This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can create an array of integers, which can result in a Memory Error and crash your program. It’s a memory hungry beast.

Q.12) We know that Python is an object-oriented language, but does it have access specifiers?
A) No. Python is a modern object-oriented language that considers the use of access specifiers (like private and public in C++) as primitive and redundant.

Q.13) Can you explain how an exception can be caught in a Python program?
A) By surrounding your code between the “try” and “except” keywords. You can either use Python’s built-in exceptions or define a new exception class. You can also raise an exception in a specific case using the raise keyword.

Q.14) How to explore a dataset in Python?
A) There are multiple commands which can help you in exploring a data set. Following are a few commands:
.info()
.describe()
.head()

Q.15) How would you test Python code?
A) Python has some cool modules to write test cases like unittest and Doctest which come bundled up with the Python standard library. You can also use tools like py.test, Hypothesis, mock, tox, and Unittest2.

Q.16) If you have to choose between a list, set, and a dictionary to store 10 million integers, what will you use? Bear in mind that you would later like to query the frequency of a number within the dataset.
A) Since we have to cater for multiple occurrences of a number, we can’t choose a set, since it doesn’t store duplicates by design.
To find the frequency of a number within a list, you will have to iterate it completely, resulting in an O(n) complexity for searching, which is inefficient.

A dictionary, however, lets you store key-value pairs. You can store a number as the key and the number of times it has been stored, as the value. This way, whenever you want to query the frequency, you can get the result in O(1). This makes a dictionary the best choice in this particular scenario.

Q.17) What do you know about the global interpreter lock?
A) The Global Interpreter Lock, aka GIL, is a mutex that prevents multiple threads to execute Python bytecode simultaneously. This lock is necessary because the memory management in CPython isn't thread-safe. This may prevent multi-threaded applications to use multiple CPUs and is, therefore, a bottleneck. However, not all the operations are concerned by the GIL, such as the IO operations or some computation done within libraries (e.g. numpy).

Q.18) What is the Dictionary?
A) Dictionary objects can be created by using curly braces {} or by calling dictionary function
- Dictionary objects are mutable objects
- Dictionary represents key-value base
- Each key-value pair of Dictionary is known as an item
- Dictionary keys must be immutable
- Dictionary values can be mutable or immutable
- Duplicate keys are not allowed but values can be duplicated
- Insertion order is not preserved
- Heterogeneous keys and heterogeneous values are allowed.

Q.19) Is there a tool to help find bugs or perform the static analysis?
A) Yes.
PyChecker is a static analysis tool that finds bugs in Python source code and warns about code complexity and style.
Pylint is another tool that checks if a module satisfies a coding standard, and also makes it possible to write plug-ins to add a custom feature.

Q.20) Python and multithreading. Is it a good idea? List some ways to get some Python code to run in a parallel way and why is it beneficial?

A) 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.

However, there are reasons to use Python's threading package in certain scenarios. 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.

Q.21) What is NumPy? Is it better than a list?

A) NumPy, a Python package, has made its place in the world of scientific computing. It can deal with large data sizes, and also has a powerful N-dimensional array object along with a set of advanced functions.

Yes, a NumPy array is better than a Python list. This is in the following ways:

- It is more compact.
- It is more convenient.
- It Is more efficiently.
- It is easier to read and write items with NumPy.

Q.22) What advantages do NumPy arrays offer over nested Python lists?

A) 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.
They have certain limitations: they don’t support “vectorized” operations like element-wise 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.
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.
NumPy array is faster and you get a lot built-in with NumPy, FFTs, convolutions, fast searching, basic statistics, linear algebra, histograms, etc.

Q.23) What are local variables and global variables in Python?
A) 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:
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.
Example:

```
a=2 #Global Variable
def add():
  b=3 #Local Variable
  c=a+b
  print(c)
add()
```

Output: 5
When you try to access the local variable outside the function add(), it will throw an error.

Q.24) What is Pandas and how is it useful for analytics?
A) Pandas is a Python library that provides highly flexible and powerful tools and high-level data structures for analysis. Pandas is an excellent tool for data analytics because it can translate highly complex operations with data into just one or two commands.
Pandas comes with a variety of built-in methods for combining, filtering, and grouping data. It also boasts time-series functionality that is closely followed by remarkable speed indicators.

Q.25) What is SciPy and how is it Useful for Data Science Work
A) SciPy is another outstanding library for scientific computing. It's based on NumPy and was created to extend its capabilities. Like NumPy, SciPy's data structure is also a multidimensional array that's implemented by NumPy.
The SciPy package contains powerful tools that help solve tasks related to integral calculus, linear algebra, probability theory, and much more.

Q.26) Python or R – Which one would you prefer for text analytics?
A) We will prefer Python because of the following reasons:
- Python would be the best option because it has Pandas library that provides easy to use data structures and high-performance data analysis tools.
- R is more suitable for machine learning than just text analysis.
- Python performs faster for all types of text analytics.

Q.27) What is a negative index, and how is it used in Python?
A) A negative index is used in Python to index a list, string, or any other container class in reverse order (from the end). Thus, [-1] refers to the last element, [-2] refers to the second-to-last element, and so on.

Q.28) Is Python fully object oriented?

A) Python does follow an object-oriented programming paradigm and has all the basic OOPs concepts such as inheritance, polymorphism, and more, with the exception of access specifiers. Python doesn’t support strong encapsulation (adding private keyword before data members) although it does have a convention that can be used for data hiding, i.e., prefixing data members with two underscores.

Q.29) What do you understand by Tkinter?
A) Tkinter is an inbuilt Python module that is used to create GUI applications. It’s Python’s standard toolkit for GUI development. Tkinter comes with Python, so there is no installation needed. We can start using it by importing it in our script.

Q.30) Can we make multi-line comments in Python?
A) Python does not have a specific syntax for including multi-line comments like other programming languages, but programmers can use triple-quoted strings (docstrings) for making multi-line comments, as when docstring is not being used as the first statement inside a method, it gets ignored by the Python parser.

Q.31) Explain the use of with statement and its syntax.
A) In Python, using the ‘with’ statement, we can open a file and close it as soon as the block of code, where ‘with’ is used, exits, without having to use the close() method.
with open(“filename”, “mode”) as file_var:

Q.32) What are negative indexes and why are they used?
A) To access an element from ordered sequences, we simply use the index of the element, which is the number of the position of that element. The index usually starts from 0, meaning that the first element has the index 0 and the second has 1, and so on.
When we use the index to access elements from the end of a list, it’s called reverse indexing. In reverse indexing, the indexing of elements start from the last element with the index number being ‘−1’. The second last element has the index ‘−2’, and so on. These indexes used in reverse indexing are called negative indexes.

Q.33) How are compile-time and run-time code checking done in Python?
A) Python has a unique way of performing compile-time and run-time code checking. A small portion checking is carried out during compile-time checking, but most of the checks such as type, name, etc are postponed until code execution. If the Python code references a user-defined function that does not exist, the code will compile successfully and the code will fail with an exception only when the code execution path references the function which does not exist.

Q.34) Which library would you prefer for plotting in Python language: Seaborn or Matplotlib?
A) Matplotlib is a python library used for plotting but it needs a lot of fine-tuning to ensure that the plots look shiny. Seaborn helps data scientists create statistically and aesthetically appealing meaningful plots. The answer to this question varies based on the requirements for plotting data.

Q.35) What is the difference between deep and shallow copy?
A) - Shallow copy is used when a new instance type gets created and it keeps the values that are copied in the new instance. Whereas, deep copy is used to store the values that are already copied.
- Shallow copy is used to copy the reference pointers just like it copies the values. These references point to the original objects and the changes made in any member of the class will also affect the original copy of it. Whereas, deep copy doesn’t copy the reference pointers to the objects. Deep copy makes the reference to an object and the new object that is pointed by some other object gets stored. The changes made in the original copy won’t affect any other copy that uses the object.
- Shallow copy allows faster execution of the program and it depends on the size of the data that is used. Whereas, deep copy makes it

slower due to making certain copies for each object that has been called.

Rapid Fire Interview Questions

Q.1) Name a few libraries in Python used for Data Analysis and Scientific computations.
A) NumPy, SciPy, Pandas, SciKit, Matplotlib, Seaborn

Q.2) What Does The "Self" Keyword Do?
A) The self is a Python keyword that represents a variable that holds the instance of an object.
In almost all the object-oriented languages, it is passed to the methods as a hidden parameter.

Q.3) Which is the standard data missing marker used in Pandas?
A) NaN

Q.4) Which Python library would you prefer to use for Data Munging?
A) Pandas

Q.5) Which python library is built on top of matplotlib and Pandas to ease data plotting?
A) Seaborn

Q.6) What is pylab?
A) A package that combines NumPy, SciPy, and Matplotlib into a single namespace.

Q.7) Name one of the most commonly used python libraries for Machine Learning?
A) SciKit-Learn

Q.8) What does _init_.py do?
A) _init_.py is an empty py file used for importing a module in a directory.

Q.9) What is a pass in Python?
A) Pass in Python signifies a no operation statement indicating that nothing is to be done.

Q.10) How can you check whether a pandas data frame is empty or not?
A) The attribute df.empty is used to check whether a data frame is empty or not.

Q.11) How will you reverse a list in Python?
A) list.reverse(): This function reverses objects of list.

Q.12) Do we need to declare variables with data types in Python?
A) No. Python is a dynamically typed language, which means that Python Interpreter automatically identifies the data type of a variable based on the type of value assigned to the variable.

Q.13) Write a command to open the file c:\hello.txt for writing.
A) f= open(“hello.txt”, “wt”)

Q.14) What are the supported data types in Python?
A) Python has five standard data types:
Numbers
Strings
Lists
Tuples
Dictionaries

Q.15) What is PEP8?
A) PEP8 is a set of coding guidelines in Python language that programmers can use to write readable code which makes it easy to use for other users.

Q.16) What is pass in Python?

A) Pass means, no-operation Python statement, or in other words it is a placeholder in a compound statement, where there should be a blank left and nothing has to be written there

Q.17) What are generators in Python?
A) The way of implementing iterators are known as generators. It is a normal function except that it yields expression in the function.

Q.18) How can you copy an object in Python?
A) To copy an object in Python, you can try copy.copy () or copy.deepcopy() for the general case. You cannot copy all objects but most of them.

Q.19) Explain how to delete a file in Python?
A) By using a command os.remove (filename) or os.unlink(filename)

Q.20) Mention the use of // operator in Python?
A) It is a Floor Division operator, which is used for dividing two operands with the result as quotient showing only digits before the decimal point. For instance, 10//5 = 2 and 10.0//5.0 = 2.0.

Q.21) What's The Process To Get The Home Directory Using '~' In Python?
A) You need to import the os module, and then just a single line would do the rest.

```
  import os
  print (os.path.expanduser('~'))
```

Output:

```
  /home/runner
```

Q.22) Is There A Switch Or Case Statement In Python? If Not Then What Is The Reason For The Same?
A) No, Python does not have a Switch statement, but you can write a Switch function and then use it.

Q.23) What Is %S In Python?

A) Python has support for formatting any value into a string. It may contain quite complex expressions.
One of the common usages is to push values into a string with the %s format specifier. The formatting operation in Python has the comparable syntax as the C function printf() has.

Q.24) Is It Mandatory For A Python Function To Return A Value?
A) It is not at all necessary for a function to return any value. However, if needed, we can use None as a return value.

Q.25) What Is The Difference Between Pass And Continue In Python?
A) The continue statement makes the loop to resume from the next iteration.
On the contrary, the pass statement instructs to do nothing, and the remainder of the code executes as usual.

Coding Based Interview Questions

Q.1) How to convert a list into a comma-separated string?
A) When we want to convert a list into a string, we can use the join() method which joins all the elements into one and returns as a string.
Example:

```
weekdays = [‘sun’,’mon’,’tue’,’wed’,’thu’,’fri’,’sat’]
listAsString = ‘,’.join(weekdays)
print(listAsString)
output: sun,mon,tue,wed,thu,fri,sat
```

Q.2) How to convert a list into a tuple?
A) By using Python <tuple()> function we can convert a list into a tuple. But we can’t change the list after turning it into tuple, because it becomes immutable.
Example:

```
weekdays = [‘sun’,’mon’,’tue’,’wed’,’thu’,’fri’,’sat’]
listAsTuple = tuple(weekdays)
print(listAsTuple)
output: (‘sun’, ‘mon’, ‘tue’, ‘wed’, ‘thu’, ‘fri’, ‘sat’)
```

Q.3) How can you create Empty NumPy Array In Python?
A) We can create Empty NumPy Array in two ways in Python,

1) import numpy
   numpy.array([])
2) numpy.empty(shape=(0,0))

Q.4) Write a program to find the sum of the digits of a number in Python?
A) Python Program to Find the Sum of the Digits of a Number

```
n=int(input(“Enter a number:”))
tot=0
while(n>0):
dig=n%10
tot=tot+dig
```

```
n=n//10
print(“The total sum of digits is:”,tot)
```
Output:
```
Enter a number:1928
The total sum of digits is: 20
```

Q.5) Write a Python Program to Check if a String is a Palindrome or Not?
A) Python Program to Check if a String is a Palindrome or Not:
```
string=raw_input(“Enter string:”)
if(string==string[::-1]):
print(“The string is a palindrome”)
else:
print(“The string isn’t a palindrome”)
```
Output:
```
Enter string:MALAYALAM
The string is a palindrome
```

Q.6) How Will You Print The Sum Of Numbers Starting From 1 To 100 (Inclusive Of Both)?
A) Python Program to Print The Sum Of Numbers Starting From 1 To 100
```
print sum(range(1,101))
#range() returns a list to the sum function containing
#all the numbers from 1 to 100. Please see that
#the range function does not include the end given (101 here).
print(sum(xrange(1, 101)))
#xrange() returns an iterator rather than a list
#which is less heavy in the memory.
```

Q.7) How do I convert between tuples and lists?
A) The function tuple(seq) converts any sequence (actually, any iterable) into a tuple with the same items in the same order.
For example, tuple([1, 2, 3]) yields (1, 2, 3) and tuple(‘abc’) yields (‘a’, ‘b’, ‘c’).

If the argument is a tuple, it does not make a copy but returns the same object, so it is cheap to call tuple() when you aren't sure that an object is already a tuple.
The function list(seq) converts any sequence or iterable into a list with the same items in the same order.
For example, list((1, 2, 3)) yields [1, 2, 3] and list('abc') yields ['a', 'b', 'c'].
If the argument is a list, it makes a copy just like seq[:] would.

Q.8) Write a one-liner that will count the number of capital letters in a file. Your code should work even if the file is too big to fit in memory.
A) This can be done in several ways, but considering the fact that we need a one-liner logic I suggest something like:

```
with open(SOME_LARGE_FILE) as fh:
    count = sum(character.isupper() for line in fh for character in line)
```

Q.9) How would you create an empty NumPy array?
A) To create an empty array with NumPy, we have two options:
a. Option 1

```
>>> import numpy
>>> numpy.array([])
```

b. Option 2

```
>>> numpy.empty(shape=(0,0))
```

Q.10) How to add a value to a python array?
A) Elements can be added to an array using the append(), extend() and insert (i,x) functions.
Example:

```
  a=arr.array('d', [1.1 , 2.1 ,3.1] )
  a.append(3.4)
  print(a)
  a.extend([4.5,6.3,6.8])
  print(a)
  a.insert(2,3.8)
  print(a)
```

Output:

```
array(‘d’, [1.1, 2.1, 3.1, 3.4])
array(‘d’, [1.1, 2.1, 3.1, 3.4, 4.5, 6.3, 6.8])
array(‘d’, [1.1, 2.1, 3.8, 3.1, 3.4, 4.5, 6.3, 6.8])
```

Q.11) How to remove values to a python array?

A) Array elements can be removed using pop() or remove() method. The difference between these two functions is that the former returns the deleted value whereas the latter does not.

Example:

```
a=arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
print(a.pop())
print(a.pop(3))
a.remove(1.1)
print(a)
```

Output:

```
4.6
3.1
array(‘d’, [2.2, 3.8, 3.7, 1.2])
```

Q.12) How will you read a random line in a file?

A) We can read a random line in a file using a module named ‘random’.

For example:

```
import random
def read_random(fname):
lines = open(fname).read().splitlines()
return random.choice(lines)
print(read_random (‘hello.txt’))
```

Q.13) Given a string, write a Python program to split strings on Uppercase characters.

A) Program to split string using re.split():

```
import re
# Initialising string
ini_str = 'GarvitArya'
# Printing Initial string
```

```
print ("Initial String", ini_str)
# Splitting on UpperCase using re
res_list = [s for s in re.split("([A-Z][^A-Z]*)", ini_str) if s]
# Printing result
print("Resultant prefix", str(res_list))
```

Q.14) Given a list of strings, write a Python program to convert the given list of strings into a space-separated string.

A) Program to convert list into string using join() function:

```
def convert(lst):
    return ' '.join(lst)
# Driver code
lst = [‘garvit’, ‘arya’]
print(convert(lst))
```

Q.15) Given a string and a boolean value, write a Python program to concatenate the string with a boolean value

A) Program to concat boolean using type casting:

```
# Initialising string and boolean value
ini_string = "Facts are"
value = True
# Concatenate using str
res = ini_string +" "+str(value)
# Printing resultant string
print ("Resultant String : ", res)
```

Q.16) Write a program to delete all occurrences of character from a string

A) Program to delete a character using string function:

```
# initializing string
test_str = "DataScienceIsFun"
# initializing removal character
rem_char = "e"
# printing original string
print("The original string is : " + str(test_str))
# Using replace()
# Deleting all occurrences of character
res = test_str.replace(rem_char, "")
# printing result
print("The string after character deletion : " + str(res))
```

Q.17) Given a list of numbers and a variable K, where K is also a number, write a Python program using Numpy module, to find the number in a list which is closest to the given number K.

A) Python program to find Closest number in a list:

```
import numpy as np
def closest(lst, K):
    lst = np.asarray(lst)
    idx = (np.abs(lst - K)).argmin()
    return lst[idx]
# Driver code
lst = [3.64, 5.2, 9.42, 9.35, 8.5, 8]
K = 9.1
print(closest(lst, K))
```

Q.18) Sometimes, while working with strings, we may have situations in which we might have more than 1 space between intermediate words in strings that are mostly unwanted. How will you remove them?

A) Program Using split() + join()

```
# initializing string
test_str = "This  is   a       Sample"
# printing original string
print("The original string is : " + test_str)
# using split() + join()
# remove additional space from string
res = " ".join(test_str.split())
# printing result
print("The strings after extra space removal : " + str(res))
```

Q.19) Given a list, write a Python program to convert the given list to dictionary such that all the odd elements become the key,

and even number elements become the value.

A) Python program to Convert a list to dictionary:

```
def Convert(lst):
    res_dct = {lst[i]: lst[i + 1] for i in range(0, len(lst), 2)}
    return res_dct
# Driver code
lst = ['a', 1, 'b', 2, 'c', 3]
print(Convert(lst))
Output: {'a': 1, 'b': 2, 'c': 3}
```

Q.20) Given a list of integers and an integer variable *K*, write a Python program to find all pairs in the list with given sum *K*.

A) Python program to find all pairs in a list of integers with given sum:

```
def findPairs(lst, K):
    res = []
    while lst:
        num = lst.pop()
        diff = K - num
        if diff in lst:
            res.append((diff, num))
    res.reverse()
    return res
# Driver code
lst = [1, 5, 3, 7, 9]
K = 12
print(findPairs(lst, K))
Output: [(5, 7), (3, 9)]
```

Q.21) While programming, sometimes, we just require a certain type of data and need to discard other. How will you Extract digits from given string?

A) Program to extract digits from a string using re package:

```
import re
# initializing string
test_string = ‘1garvit2arya3’
```

```
# printing original strings
print("The original string : " + test_string)
# using re
# Extract digit string
res = re.sub("\D", "", test_string)
# print result
print("The digits string is : " + str(res))
```
Output :
```
The original string : 1garvit2arya3
The digits string is : 123
```

Q.22) Write a program to Split a string on last occurrence of delimiter

A) Python3 code to demonstrate Split on last occurrence of delimiter using rsplit()
```
# initializing string
test_string = "data, is, good, better, and best"
# printing original string
print("The original string : " + str(test_string))
# using rsplit()
# Split on last occurrence of delimiter
res = test_string.rsplit(', ', 1)
# print result
print("The post-split list at the last comma : " + str(res))
```
Output :
```
The original string : data, is, good, better, and best
The post-split list at the last comma : ['data, is, good, better', 'and best']
```

Q.23) How will you Get the string after occurrence of a given substring

A) The partition function can be used to perform this task in which we just return the part of partition occurring after the partition word.
```
# initializing string
test_string = "Python is best for data"
# initializing split word
spl_word = 'best'
```

```
# printing original string
print("The original string : " + str(test_string))
# printing split string
print("The split string : " + str(spl_word))
# using partition()
# Get String after substring occurrence
res = test_string.partition(spl_word)[2]
# print result
print("String after the substring occurrence : " + res)
```

Q.24) Given a string, write a Python program to check if the string is a valid email address or not.

A) Python program to validate an Email :

```
import re
# Make a regular expression
# for validating an Email
regex = '^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$'
# Define a function for
# for validating an Email
def check(email):
    # pass the regular expression
    # and the string in search() method
    if(re.search(regex,email)):
        print("Valid Email")
    else:
        print("Invalid Email")
# Driver Code
if __name__ == '__main__' :
    # Enter the email
    email = "garvit@gmail.com"
    # calling run function
    check(email)
    email = "garvit.com"
    check(email)
```

Q.25) Sometimes, during the string manipulation, we are into a problem where we need to pad or add leading zeroes to the

string as per the requirements. How will you achieve this?

A) Program to add leading zeros:

```
# initializing string
test_string = ‘GAP’
# printing original string
print("The original string : " + str(test_string))
# No. of zeros required
N = 4
# using rjust()
# adding leading zero
res = test_string.rjust(N + len(test_string), '0')
# print result
print("The string after adding leading zeros : " + str(res))
```

```
The original string : GAP
The string after adding leading zeros : 0000GAP
```

Python Cheat Sheet

Types:

```
a = 2          # integer
b = 5.0        # float
c = 8.3e5      # exponential
d = 1.5 + 0.5j  # complex
e = 4 > 5      # boolean
f = 'word'     # string
```

Lists:

```
a = ['red', 'blue', 'green']     # manually initialization
b = list(range(5))              # initialize from iterable
c = [nu**2 for nu in b]          # list comprehension
d = [nu**2 for nu in b if nu<3]  # conditioned list comprehension
e = c[0]                        # access element
f = c[1:2]                      # access a slice of the list
g = c[-1]                       # access last element
h = ['re', 'bl'] + ['gr']        # list concatenation
i = ['re'] * 5                  # repeat a list
['re', 'bl'].index('re')         # returns index of 're'
a.append('yellow')               # add new element to end of list
a.extend(b)                     # add elements from 'b' to 'a'
a.insert(1, 'yellow')            # insert in specified position
're' in ['re', 'bl']            # true if 're' in list
'fi' not in ['re', 'bl']        # true if 'fi' not in list
sorted([3, 2, 1])               # returns sorted list
a.pop(2)                        # remove & return item at index
```

Dictionaries:

```
a = {'red': 'rouge', 'blue': 'bleu'} # dictionary
b = a['red']                     # translate item
'red' in a                       # true if ‘a’ contains 'red'
c = [value for key, value in a.items()]  # loop through contents
a.setdefault('extra', []).append('cyan') # init key with default
```

```
a.update({'green': 'vert'}) # update ‘a’ by data from another one
a.keys()                # get list of keys
a.values()              # get list of values
a.items()               # get list of key-value pairs
del a['red']            # delete key and associated value
a.pop('blue')           # remove specified key & return value
```

Sets:

```
a = {1, 2, 3}           # initialize manually
b = set(range(5))       # initialize from iterable
a.add(13)               # add new element to set
a.discard(13)           # discard element from set
a.update([21, 22, 23])  # update set with elements from iterable
a.pop()                 # remove and return set element
2 in {1, 2, 3}          # true if 2 in set
5 not in {1, 2, 3}      # true if 5 not in set
a.issubset(b)           # test whether every element in a is in b
a <= b                  # issubset in operator form
a.issuperset(b)         # test whether every element in b is in a
a >= b                  # issuperset in operator form
a.intersection(b)       # return the intersection of two sets
a.difference(b)         # return the difference of two sets
a - b                   # difference in operator form
a.union(b)              # return the union of sets as a new set
c = frozenset()         # the same as set but immutable
```

Strings:

```
a = 'red'                     # assignment
char = a[2]                   # access individual characters
'red ' + 'blue'               # string concatenation
'1, 2, three'.split(',')      # split string into list
'.'.join(['1', '2', 'three'])  # concatenate list into string
```

Operators:

```
a = 2              # assignment
a += 1 (*=, /=)    # change and assign
```

```
3 + 2           # addition
3 / 2           # integer (py2) or float (py3) division
3 // 2          # integer division
3 * 2           # multiplication
3 ** 2          # exponent
3 % 2           # remainder
abs(a)          # absolute value
1 == 1          # equal
2 > 1           # larger
2 < 1           # smaller
1 != 2          # not equal
1 != 2 and 2 < 3  # logical AND
1 != 2 or 2 < 3   # logical OR
not 1 == 2      # logical NOT
'a' in b        # test if a is in b
```

Control Flow:

```
# if/elif/else
a, b = 1, 2
if a + b == 3:
   print('True')
elif a + b == 1:
   print('False')
else:
   print('?')

# for
a = ['red', 'blue', 'green']
for color in a:
   print(color)

# while
number = 1
while number < 10:
   print(number)
```

```
    number += 1

# break
number = 1
while True:
    print(number)
    number += 1
    if number > 10:
        break

# continue
for i in range(20):
    if i % 2 == 0:
        continue
    print(i)
```

Functions, Classes, Generators, Decorators:

```
# Function groups code statements and possibly
# returns a derived value
def myfunc(a1, a2):
    return a1 + a2
x = myfunc(a1, a2)

# Class groups attributes (data)
# and associated methods (functions)
class Point(object):
    def __init__(self, x):
        self.x = x
    def __call__(self):
        print(self.x)
x = Point(3)

# Generator iterates without
# creating all values at once
def firstn(n):
    num = 0
    while num < n:
        yield num
        num += 1
x = [i for i in firstn(10)]

# Decorator can be used to modify
# the behaviour of a function
class myDecorator(object):
    def __init__(self, f):
        self.f = f
    def __call__(self):
        print("call")
        self.f()

@myDecorator
def my_funct():
    print('func')
my_funct()
```

debugger:

```
n              # execute next line
b 42           # set breakpoints in the main file at line 42
b myfile.py:42  # set breakpoint in 'myfile.py' at line 42
c              # continue execution
l              # show current position in the code
p data         # print the 'data' variable
pp data        # pretty print the 'data' variable
s              # step into subroutine
a              # print arguments that a function received
pp locals()    # show all variables in local scope
pp globals()   # show all variables in global scope
```

NumPy Cheat Sheet (import numpy as np)

array initialization:

```
np.array([2, 3, 4])          # direct initialization
np.empty(8, dtype=np.float32) # single precision array of size 8
np.zeros(200)                # initialize 200 zeros
np.ones((3,3),dtype=np.int32) # 3 x 3 integer matrix with ones
np.eye(200)                  # ones on the diagonal
np.zeros_like(a)             # array with zeros and shape of a
np.linspace(0., 10., 100)    # 100 points from 0 to 10
np.arange(0, 100, 2)         # points from 0 to <100 with step 2
np.logspace(-5, 2, 100)      # 100 log-spaced from 1e-5 -> 1e2
np.copy(a)                   # copy array to new memory
```

indexing:

```
a = np.arange(100)     # initialization with 0 - 99
a[:3] = 0              # set the first three indices to zero
a[2:5] = 1             # set indices 2-4 to 1
a[:-3] = 2             # set all but last 3 elements to 2
a[start:stop:step]     # general form of indexing/slicing
a[None, :]             # transform to column vector
a[[1, 1, 3, 8]]        # return array with values of the indices
a = a.reshape(10, 10)  # transform to 10 x 10 matrix
a.T                    # return transposed view
a[a < 2]               # values with elementwise condition
```

array properties and operations:

```
a.shape              # a tuple with the lengths of each axis
len(a)               # length of axis 0
a.ndim               # number of dimensions (axes)
a.sort(axis=1)       # sort array along axis
a.flatten()          # collapse array to one dimension
a.conj()             # return complex conjugate
a.astype(np.int16)   # cast to integer
```

```
a.tolist()          # convert array to list
np.argmax(a, axis=1) # return index of maximum along a given axis
np.cumsum(a)        # return cumulative sum
np.any(a)           # True if any element is True
np.all(a)           # True if all elements are True
np.argsort(a,axis=1) # return sorted index array along axis
np.where(cond)      # return indices where cond is True
np.where(cond, x, y) # return from x or y depending on cond
```

Boolean arrays:

```
a < 2               # returns array with boolean values
(a < 2) & (b > 10)  # element wise logical and
(a < 2) | (b > 10)  # element wise logical or
~a                  # invert boolean array
```

elementwise operations and math functions:

```
a * 5           # multiplication with scalar
a + 5           # addition with scalar
a + b           # addition with array b
a / b           # division with b
np.exp(a)       # exponential (complex and real)
np.power(a, b)  # a to the power b
np.sin(a)       # sine
np.cos(a)       # cosine
np.arctan2(a, b) # arctan(a/b)
np.arcsin(a)    # arcsin
np.radians(a)   # degrees to radians
np.degrees(a)   # radians to degrees
np.var(a)       # variance of array
np.std(a,axis=1) # standard deviation
```

inner/ outer products:

```
np.dot(a, b)            # inner product: a_mi b_in
np.sum(a, axis=1)       # sum over axis 1
np.abs(a)               # return absolute values
a[None, :] + b[:, None] # outer sum
```

```
a[None, :] * b[:, None]     # outer product
np.outer(a, b)              # outer product
np.sum(a * a.T)             # matrix norm
```

rounding:

```
np.ceil(a)   # rounds to the nearest upper int
np.floor(a)  # rounds to the nearest lower int
np.round(a)  # rounds to the nearest int
```

random variables:

```
from np.random import normal, seed, rand, uniform, randint
normal(loc=0, scale=2, size=100)  # 100 normal distributed
seed(23032)                       # resets the seed value
rand(200)                         # 200 random numbers in [0, 1)
uniform(1, 30, 200)               # 200 random numbers in [1, 30)
randint(1, 16, 300)               # 300 random integers in [1, 16)
```

Matplotlib Cheat Sheet (import matplotlib.pyplot as plt)

figures and axes:

fig = plt.figure(figsize=(5, 2)) # initialize figure
fig.savefig('out.png')   # save png image
fig, axes = plt.subplots(5, 2, figsize=(5, 5)) # fig and 5 x 2 nparray of axes
ax = fig.add_subplot(3, 2, 2) # add 2nd subplot in a 3 x 2 grid
ax = plt.subplot2grid((2, 2), (0, 0), colspan=2) # multi column/row axis
ax = fig.add_axes([left, bottom, width, height]) # add custom axis

figures and axes properties:

fig.suptitle('title')   # big figure title
fig.subplots_adjust(bottom=0.1, right=0.8, top=0.9, wspace=0.2, hspace=0.5) # adjust subplot positions
fig.tight_layout(pad=0.1, h_pad=0.5, w_pad=0.5, rect=None) # adjust subplots to fit into fig
ax.set_xlabel('xbla')   # set xlabel
ax.set_ylabel('ybla')   # set ylabel
ax.set_xlim(1, 2)   # sets x limits
ax.set_ylim(3, 4)   # sets y limits
ax.set_title('blabla')   # sets the axis title
ax.set(xlabel='bla')   # set multiple parameters at once
ax.legend(loc='upper center') # activate legend
ax.grid(True, which='both')   # activate grid
bbox = ax.get_position()   # returns the axes bounding box
bbox.x0 + bbox.width   # bounding box parameters

plotting routines:

ax.plot(x,y, '-o', c='red', lw=2, label='bla') # plots a line
ax.scatter(x,y, s=20, c=color)   # scatter plot
ax.pcolormesh(xx, yy, zz, shading='gouraud')   # fast colormesh
ax.colormesh(xx, yy, zz, norm=norm)   # slow colormesh

```
ax.contour(xx, yy, zz, cmap='jet')            # contour lines
ax.contourf(xx, yy, zz, vmin=2, vmax=4)       # filled contours
n, bins, patch = ax.hist(x, 50)               # histogram
ax.imshow(matrix, origin='lower', extent=(x1, x2, y1, y2)) # show
image
ax.specgram(y, FS=0.1, noverlap=128, scale='linear') # plot a
spectrogram
ax.text(x, y, string, fontsize=12, color='m') # write text
```

Pandas Cheat Sheet (import pandas as pd)

Data structures:

```
s = pd.Series(np.random.rand(10), index=range(10)) # series
index = pd.date_range("13/06/2016", periods=100)   # time index
df = pd.DataFrame(np.zeros((1000, 3)), index=index, columns=["A",
"B", "C"])   # DataFrame creation
```

DataFrame:

```
df = pd.read_csv("file.csv") # read and load CSV in a DF
raw = df.values              # get raw data out of DF object
cols = df.columns            # get list of columns headers
df.dtypes                    # get data types of all columns
df.head(5)                   # get first 5 rows
df.describe()                # get basic statistics for columns
df.index                     # get index column range
```

column slicing:

```
df.col_name           # select column values as a series
df[['col_name']]      # select column values as a DF
df.loc[:, 'col_name']  # select column values as a series
df.loc[:,['col_name']] # select column values as a DF
df.iloc[:, 0]         # select by column index
df.iloc[:, [0]]       # select by column index, but as a DF
df.ix[:, 'col_name']   # hybrid approach with column name
df.ix[:, 0]           # hybrid approach with column index
```

row slicing:

```
print(df[:2])         # print first 2 rows of the DF
df.iloc[0:2, :]       # select first 2 rows of the DF
df.loc[0:2,'col_name'] # select first 3 rows of the DF
df.loc[0:2, ['c1', 'c3']] # 1st 3 rows of given columns
df.iloc[0:2,0:2]      # select the 1st 3 rows 7 1st 3 columns
```

Dicing:

```
df[ df.col_name < 7 ]  # select all rows where col_name < 7
df[ (df.c1 < 7) & (df.c2 == 0) ]  # combine multiple conditionals using
bitwise operator
df[df.recency < 7] = -100         # writing to slice
```