52 Python Developer Interview Questions-Answers

Watch Full Video On Youtube:
https://youtu.be/YeupGcOW-3k

Connect with me:
Youtube: https://www.youtube.com/c/nitmantalks
Instagram: https://www.instagram.com/nitinmangotra/
LinkedIn: https://www.linkedin.com/in/nitin-mangotra-9a075a149/
Facebook: https://www.facebook.com/NitManTalks/
Twitter: https://twitter.com/nitinmangotra07/
Telegram: https://t.me/nitmantalks/

NITIN MANGOTRA

1. Difference Between List and Tuple

LIST

1. Lists are **mutable**
2. List is a container to contain different types of objects and is used to iterate objects.
3. Syntax Of List

   list = ['a', 'b', 'c', 1,2,3]

4. List iteration is slower
5. Lists consume more memory
6. Operations like insertion and deletion are better performed.

Tuple

1. Tuples are **immutable**
2. Tuple is also similar to list but contains immutable objects.
3. Syntax Of Tuple

   tuples = ('a', 'b', 'c', 1, 2)

4. Tuple processing is faster than List.
5. Tuple consume less memory
6. Elements can be accessed better.

NITIN MANGOTRA

2. What is Decorator? Explain With Example.

A Decorator is just a function that takes another function as an argument, add some kind of functionality and then returns another function.

All of this without altering the source code of the original function that you passed in.

NITIN MANGOTRA

2. What is Decorator? Explain With Example.

```
def decorator_func(func):
        def wrapper_func():
                print("wrapper_func Worked")
                return func()
        print("decorator_func worked")
        return wrapper_func



def show():
        print("Show Worked")
decorator_show = decorator_func(show)
decorator_show()


#Alternative
@decorator_func
def display():
        print('display
                worked')
display()
```

Output:
decorator_func worked
wrapper_func Worked
Show Worked

Output:
decorator_func worked
wrapper_func Worked
display worked

NITIN MANGOTRA

3. Difference Between List and Dict Comprehension

List Comprehension

Syntax:

[expression for item in iterable if conditional]

Example:
Common Way:

```
l = []
for i in range(10):
   if i%2:
      l.append(i)
print(l)
```

Using List Comprehension:

```
ls = [i for i in range(10) if i%2]
print(ls)
```

Output:
[1, 3, 5, 7, 9]

Dict Comprehension

Syntax :

{key:value for (key,value) in iterable if conditional}

Example:
Common Way:

```
d = {}
for i in range(1,10):
        sqr = i*i
        d[i] = i*i
print(d)
```

Using Dict Comprehension:

```
d1={n:n*n for n in range(1,10)}
print (d1)
```

Output:
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

4. How Memory Managed In Python?

- Memory management in Python involves a **private heap** containing all Python objects and data structures. Interpreter takes care of Python heap and that the programmer has no access to it.
- The allocation of heap space for Python objects is done by **Python memory manager**. The core API of Python provides some tools for the programmer to code reliable and more robust program.
- Python also has a build-in garbage collector which recycles all the unused memory. When an object is no longer referenced by the program, the heap space it occupies can be freed. The garbage collector determines objects which are no longer referenced by the program frees the occupied memory and make it available to the heap space.
- The gc module defines functions to enable /disable garbage collector:

- **gc.enable()** -Enables automatic garbage collection.
- **gc.disable()** - Disables automatic garbage collection.

Source: https://www.careerride.com/python-memory-management.aspx

5. Difference Between Generators And Iterators

GENERATOR

- Generators are iterators which can execute only once.
- Generator uses "yield" keyword.
- Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop.
- Every generator is an iterator.

EXAMPLE:

```
def sqr(n):
   for i in range(1, n+1):
      yield i*i
a = sqr(3)
print(next(a))
print(next(a))
print(next(a))
```

Output:
1
4
9

ITERATOR

- An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc.
- Iterators are used mostly to iterate or convert other objects to an iterator using iter() function.
- Iterator uses iter() and next() functions.
- Every iterator is not a generator.

Example:

```
iter_list = iter(['A', 'B', 'C'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Output:
A
B
C

6. What is 'init' Keyword In Python?

___init___.py file

The ___init___.py file lets the Python interpreter know that a directory contains code for a Python module. It can be blank. Without one, you cannot import modules from another folder into your project.

The role of the ___init___.py file is similar to the ___init___ function in a Python class. The file essentially the constructor of your package or directory without it being called such. It sets up how packages or functions will be imported into your other files.

___init___() function

The ___init___ method is similar to **constructors** in C++ and Java. Constructors are used to initialize the object's state.

```
# A Sample class with init method
class Person:
    # init method or constructor
    def ___init___(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Nitin')
p.say_hi()
```

Output:

Hello, my name is Nitin

NITIN MANGOTRA

7. Difference Between Modules and Packages in Python

Module

The module is a simple Python file that contains collections of functions and global variables and with having a .py extension file. It is an executable file and to organize all the modules we have the concept called **Package** in Python.

A module is a single file (or files) that are imported under one import and used.
E.g.

```
import <my_module>

Import numpy
```

NITIN MANGOTRA

7. Difference Between Modules and Packages in Python

Package

The package is a simple directory having collections of modules. This directory contains Python modules and also having ___init___.py file by which the interpreter interprets it as a Package. The package is simply a namespace. The package also contains sub-packages inside it.

A package is a collection of modules in directories that give a package hierarchy.
E.g

```
from my_package.abc import a
```

NITIN MANGOTRA

8. Difference Between Range and Xrange?

Parameters	Range()	Xrange()
Return type	It returns a list of integers.	It returns a generator object.
Memory Consumption	Since range() returns a list of elements, it takes more memory.	In comparison to range(), it takes less memory.
Speed	Its execution speed is slower.	Its execution speed is faster.
Python Version	Python 2, Python 3	xrange no longer exists.
Operations	Since it returns a list, all kinds of arithmetic operations can be performed.	Such operations cannot be performed on xrange().

NITIN MANGOTRA

9. What are Generators. Explain it with Example.

- Generators are iterators which can execute only once.
- Every generator is an iterator.
- Generator uses “yield” keyword.
- Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop

Example:

```
def sqr(n):
   for i in range(1, n+1):
      yield i*i
a = sqr(3)

print("The square are : ")
print(next(a))
print(next(a))
print(next(a))
```

Output:

```
The square are :
1
4
9
```

NITIN MANGOTRA

10. What are in-built Data Types in Python OR Explain Mutable and Immutable Data Types

A first fundamental distinction that Python makes on data is about whether or not the value of an object changes.
If the value can change, the object is called **mutable**, while if the value cannot change, the object is called **immutable**.

DataType	Mutable Or Immutable?
Boolean (bool)	Immutable
Integer (int)	Immutable
Float	Immutable
String (str)	Immutable
tuple	Immutable
frozenset	Immutable
list	Mutable
set	Mutable
dict	Mutable

NITIN MANGOTRA

11. Explain Ternary Operator in Python?

The syntax for the Python ternary statement is as follows:

```
[if_true] if [expression] else [if_false]
```

Ternary Operator Example:

```
age = 25
discount = 5 if age < 65 else 10
print(discount)
```

NITIN MANGOTRA

12. What is Inheritance In Python

In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class.

In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name.

```
Class A(B):
```

```
class A:
   def display(self):
      print("A Display")

class B(A):
   def show(self):
      print("B Show")
d = B()
d.show()
d.display()
```

Output:
B Show
A Display

NITIN MANGOTRA

13. Difference Between Local and Global Variable in Python

Local Variable	Global Variable
It is declared inside a function.	It is declared outside the function.
If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Parameters passing is required for local variables to access the value in other function	Parameters passing is not necessary for a global variable as it is visible throughout the program
When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Local variables can be accessed with the help of statements, inside a function in which they are declared.	You can access global variables by any statement in the program.
It is stored on the stack unless specified.	It is stored on a fixed location decided by the compiler.

NITIN MANGOTRA

Source: https://www.guru99.com/local-vs-global-variable.html

14. Explain Break, Continue and Pass Statement

- A **break** statement, when used inside the loop, will terminate the loop and exit. If used inside nested loops, it will break out from the current loop.
- A **continue** statement will stop the current execution when used inside a loop, and the control will go back to the start of the loop.
- A **pass** statement is a null statement. When the Python interpreter comes across the pass statement, it does nothing and is ignored.

Break Statement Example

```
for i in range(10):
   if i == 7:
      break
   print( i, end = ",")
```

Output:
0,1,2,3,4,5,6,

Continue Statement Example

```
for i in range(10):
   if i == 7:
      continue
   print( i, end = ",")
```

Output:
0,1,2,3,4,5,6,8,9,

Pass Statement Example

```
def my_func():
   print('pass inside function')
   pass
my_func()
```

Output:
pass inside function

NITIN MANGOTRA

15. What is 'self' Keyword in python?

The **'self'** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

```
class Person:
   def __init__(self, name, age):
      self.name = name
      self.age = age

   def info(self):
      print(f"My name is {self.name}. I am {self.age} years old.")

c = Person("Nitin",23)
c.info()
```

Output:
My name is Nitin. I am 23 years old.

NITIN MANGOTRA

16. Difference Between Pickling and Unpickling?

Pickling:

In python, the pickle module accepts any Python object, transforms it into a string representation, and dumps it into a file by using the dump function. This process is known as **pickling**. The function used for this process is pickle.dump()

Unpickling:

The process of retrieving the original python object from the stored string representation is called **unpickling**.
The function used for this process is pickle.load()

- They are inverses of each other.
- **Pickling**, also called **serialization**, involves converting a Python object into a series of bytes which can be written out to a file.
- **Unpicking**, or **de-serialization**, does the opposite–it converts a series of bytes into the Python object it represents.

NITIN MANGOTRA

17. Explain Function of List, Set, Tuple And Dictionary?

Functions Of List

- **sort():** Sorts the list in ascending order.
- **append():** Adds a single element to a list.
- **extend():** Adds multiple elements to a list.
- **index():** Returns the first appearance of the specified value.
- **max(list):** It returns an item from the list with max value.
- **min(list):** It returns an item from the list with min value.
- **len(list):** It gives the total length of the list.
- **list(seq)**: Converts a tuple into a list.
- **cmp(list1, list2):** It compares elements of both lists list1 and list2.
- **type(list):** It returns the class type of an object.

NITIN MANGOTRA

17. Explain Function of List, Set, Tuple And Dictionary?

Functions Of Tuple

- **cmp(tuple1, tuple2):** Compares elements of both tuples.
- **len():** total length of the tuple.
- **max():** Returns item from the tuple with max value.
- **min():** Returns item from the tuple with min value.
- **tuple(seq):** Converts a list into tuple.
- **sum():** returns the arithmetic sum of all the items in the tuple.
- **any():** If even one item in the tuple has a Boolean value of True, it returns True. Otherwise, it returns False.
- **all():** returns True only if all items have a Boolean value of True. Otherwise, it returns False.
- **sorted():** a sorted version of the tuple.
- **index():** It takes one argument and returns the index of the first appearance of an item in a tuple
- **count():** It takes one argument and returns the number of times an item appears in the tuple.

NITIN MANGOTRA

17. Explain Function of List, Set, Tuple And Dictionary?

Functions Of Dictionary

- **clear():** Removes all the elements from the dictionary
- **copy():** Returns a copy of the dictionary
- **fromkeys():** Returns a dictionary with the specified keys and value
- **get():** Returns the value of the specified key
- **items():** Returns a list containing a tuple for each key value pair
- **keys():** Returns a list containing the dictionary's keys
- **pop():** Removes the element with the specified key
- **popitem():** Removes the last inserted key-value pair
- **setdefault()**: Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
- **update():** Updates the dictionary with the specified key-value pairs
- **values():** Returns a list of all the values in the dictionary
- **cmp():** compare two dictionaries

NITIN MANGOTRA

17. Explain Function of List, Set, Tuple And Dictionary?

Functions Of Set

- **add():** Adds an element to the set
- **clear():** Removes all the elements from the set
- **copy():** Returns a copy of the set
- **difference():** Returns a set containing the difference between two or more sets
- **difference_update():** Removes the items in this set that are also included in another, specified set
- **discard():** Remove the specified item
- **intersection():** Returns a set, that is the intersection of two or more sets
- **intersection_update():** Removes the items in this set that are not present in other, specified set(s)
- **isdisjoint():** Returns whether two sets have a intersection or not
- **issubset():** Returns whether another set contains this set or not
- **issuperset():** Returns whether this set contains another set or not
- **pop():** Removes an element from the set
- **remove():** Removes the specified element
- **symmetric_difference():** Returns a set with the symmetric differences of two sets
- **symmetric_difference_update():** inserts the symmetric differences from this set and another
- **union():** Return a set containing the union of sets
- **update():** Update the set with another set, or any other iterable

NITIN MANGOTRA

18. What are Python Iterators?

- An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc.
- Iterators are used mostly to iterate or convert other objects to an iterator using iter() function.
- Iterator uses iter() and next() functions.
- Every iterator is not a generator.

Example:

```
iter_list = iter(['A', 'B', 'C'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Output:
A
B
C

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **int()** - Converts any data type into an integer.

Example:

```
a = '100'
d = int(a)
print(d)
print(type(d))
```

Output:
100
<class 'int'>

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **float()** - Returns A floating point number from a number or string

Example:
```
a = '100'
d = float(a)
print(d)
print(type(d))
```

Output:
```
100.0
<class 'float'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **oct()** - Returns its octal representation in a string format.

Example:
```
a = 100
d = oct(a)
print(d)
print(type(d))
```

Output:
```
00144
<class 'str'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **hex()** - Convert the integer into a suitable hexadecimal form for the number of the integer.

Example:
```
a = 100
d = hex(a)
print(d)
print(type(d))
```

Output:
```
0x64
<class 'str'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **ord()** - Returns the integer of the Unicode point of the character in the Unicode case or the byte value in the case of an 8-bit argument.

Example:
```
a = 'A'
d = ord(a)
print(d)
print(type(d))
```

Output:
```
65
<class 'int'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **chr()** - Returns the character (string) from the integer (represents unicode code point of the character).

Example:
```
a = 100
d = chr(a)
print(d)
print(type(d))
```

Output:
```
d
<class 'str'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **eval()** - Parses the expression argument and evaluates it as a python expression.

Example:
```
a = '100+2+3'
d = eval(a)
print(d)
print(type(d))
```

Output:
```
105
<class 'int'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **str()** - Convert a value (integer or float) into a string.

Example:
```
a = 100
d = str(a)
print(d)
print(type(d))
```

Output:
```
100
<class 'str'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **repr()** - Returns the string representation of the value passed to eval function by default. For the custom class object, it returns a string enclosed in angle brackets that contains the name and address of the object by default.

Example:
```
a = 100
d = repr(a)
print(d)
print(type(d))
```

Output:
```
100
<class 'str'>
```

NITIN MANGOTRA

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- **int**() - Converts any data type into an integer.
- **float**() - Returns A floating point number from a number or string
- **oct**() - Returns its octal representation in a string format.
- **hex**() - Convert the integer into a suitable hexadecimal form for the number of the integer.
- **ord**() - Returns the integer of the Unicode point of the character in the Unicode case or the byte value in the case of an 8-bit argument.
- **chr**(number) - Returns the character (string) from the integer (represents unicode code point of the character).
- **eval**() - Parses the expression argument and evaluates it as a python expression.
- **str**() - Convert a value (integer or float) into a string.
- **repr**() - Returns the string representation of the value passed to eval function by default. For the custom class object, it returns a string enclosed in angle brackets that contains the name and address of the object by default.

NITIN MANGOTRA

20. What does *args and **kwargs mean? Expain

When you are not clear how many arguments you need to pass to a particular function, then we use ***args** and ****kwargs**.

The ***args** keyword represents a varied number of arguments. It is used to add together the values of multiple arguments

The ****kwargs** keyword represents an arbitrary number of arguments that are passed to a function. **kwargs keywords are stored in a dictionary. You can access each item by referring to the keyword you associated with an argument when you passed the argument.

*args Python Example:

```
def sum(*args):
        total = 0
        for a in args:
                total = total + a
        print(total)

sum(1,2,3,4,5)
```

Output:
15

**Kwargs Python Example

```
def show(**kwargs):
        print(kwargs)

show(A=1,B=2,C=3)
```

Output:
{'A': 1, 'B': 2, 'C': 3}

NITIN MANGOTRA

21. What is "Open" and "With" statement in Python?

- Both Statements are used in case of file handling.
- With the “With” statement, you get better syntax and exceptions handling.

```
f = open("nitin.txt")
content = f.read()
print(content)
f.close()
```

NITIN MANGOTRA

21. What is "Open" and "With" statement in Python?

- Both Statements are used in case of file handling.
- With the “With” statement, you get better syntax and exceptions handling.

```
f = open("nitin.txt")
content = f.read()
print(content)
f.close()
```

```
with open("nitin.txt") as f:
        content = f.read()
        print(content)
```

NITIN MANGOTRA

22. Different Ways To Read And Write In A File In Python?

Syntax of Python open file function:

```
file_object  = open("filename", "mode")
```

- **Read Only ('r')** : Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
- **Read and Write ('r+')** : Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
- **Write Only ('w')** : Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists
- **Write and Read ('w+')** : Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
- **Append Only ('a')** : Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
- **Append and Read ('a+')** : Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
- **Text mode ('t'):** meaning \n characters will be translated to the host OS line endings when writing to a file, and back again when reading.
- **Exclusive creation ('x'):** File is created and opened for writing – but only if it doesn't already exist. Otherwise you get a FileExistsError.
- **Binary mode ('b'):** appended to the mode opens the file in binary mode, so there are also modes like 'rb', 'wb', and 'r+b'.

NITIN MANGOTRA

23. What is Pythonpath?

PYTHONPATH is an environment variable which you can set to add additional directories where python will look for modules and packages

The 'PYTHONPATH' variable holds a string with the name of various directories that need to be added to the sys.path directory list by Python.

The primary use of this variable is to allow users to import modules that are not made installable yet.

NITIN MANGOTRA

24. How Exception Handled In Python?

Try: This block will test the exceptional error to occur.

Except: Here you can handle the error.

Else: If there is no exception then this block will be executed.

Finally: Finally block always gets executed either exception is generated or not.

```
try:
    # Some Code....!

except:
    # Optional Block
    # Handling of exception (if required)

else:
    # Some code .....
    # execute if no exception

finally:
    # Some code .....(always executed)
```

NITIN MANGOTRA

25. Difference Between Python 2.0 & Python 3.0

Basis of comparison	Python 3	Python 2

Syntax	def main():
	print("Hello World!")
	if ___name___== "___main___":
	main()
		def main():
		print "Hello World!"
		if ___name___== "___main___":
		main()

Release Date	2008	2000
Function print	print ("hello")	print "hello"
Division of Integers	Whenever two integers are divided, you get a float value	When two integers are divided, you always provide integer value.
Unicode	In Python 3, default storing of strings is Unicode.	To store Unicode string value, you require to define them with "u".
Syntax	The syntax is simpler and easily understandable.	The syntax of Python 2 was comparatively difficult to understand.

https://www.guru99.com/python-2-vs-python-3.html

25. Difference Between Python 2.0 & Python 3.0

Basis of comparison	Python 3	Python 2
Rules of ordering Comparisons	In this version, Rules of ordering comparisons have been simplified.	Rules of ordering comparison are very complex.
Iteration	The new Range() function introduced to perform iterations.	In Python 2, the xrange() is used for iterations.
Exceptions	It should be enclosed in parenthesis.	It should be enclosed in notations.
Leak of variables	The value of variables never changes.	The value of the global variable will change while using it inside for-loop.
Backward compatibility	Not difficult to port python 2 to python 3 but it is never reliable.	Python version 3 is not backwardly compatible with Python 2.
Library	Many recent developers are creating libraries which you can only use with Python 3.	Many older libraries created for Python 2 is not forward-compatible.

NITIN MANGOTRA

https://www.guru99.com/python-2-vs-python-3.html

26. What is 'PIP' In Python

Python pip is the package manager for Python packages. We can use pip to install packages that do not come with Python.

The basic syntax of pip commands in command prompt is:

```
pip 'arguments'
```

```
Pip install <package_name>
```

NITIN MANGOTRA

27. Where Python Is Used?

- Web Applications
- Desktop Applications
- Database Applications
- Networking Application
- Machine Learning
- Artificial Intelligence
- Data Analysis
- IOT Applications
- Games and many more...!

NITIN MANGOTRA

28. How to use F String and Format or Replacement Operator?

```
#How To Use f-string
name = 'Nitin'
role = 'Python Developer'
print(f"Hello, My name is {name} and I'm {role}")
```

Output:
Hello, My name is Nitin and I'm Python Developer

```
#How To Use format Operator
name = 'Nitin'
role = 'Python Developer'
print(("Hello, My name is {} and I'm {}").format(name,role))
```

Output:
Hello, My name is Nitin and I'm Python Developer

NITIN MANGOTRA

29. How to Get List of all keys in a Dictionary?

Case - 1,2: Using List

```
dct = {'A': 1, 'B': 2, 'C': 3}
all_keys = list(dct.keys())
print(all_keys)
```

Shortcut for Above Code:

```
dct = {'A': 1, 'B': 2, 'C': 3}
all_keys = list(dct)
print(all_keys)
```

Output :
['A', 'B', 'C']

NITIN MANGOTRA

29. How to Get List of all keys in a Dictionary?

Case - 3,4: Using Iterable Unpacking Operator

```
d = {'A': 1, 'B': 2, 'C': 3}
x = [*d.keys()]
print(x)
```

Shortcut For Above Code:

```
d = {'A': 1, 'B': 2, 'C': 3}
x = [*d]
print(x)
```

Output :
['A', 'B', 'C']

NITIN MANGOTRA

29. How to Get List of all keys in a Dictionary?

Case - 5: Using Keys() Function

```
d = {'A': 1, 'B': 2, 'C': 3}
x = d.keys()
print([k for k in x])
```

Output :
['A', 'B', 'C']

NITIN MANGOTRA

29. How to Get List of all keys in a Dictionary?

Case - 6,7: Using Iterable Unpacking Operator

```
d = {'A': 1, 'B': 2, 'C': 3}
*x, = d.keys()
print(x)
```

Shortcut For Above Code:

```
d = {'A': 1, 'B': 2, 'C': 3}
*x, = d
print(x)
```

Output :
['A', 'B', 'C']

NITIN MANGOTRA

30. Difference Between Abstraction and Encapsulation.

Abstraction	Encapsulation
Abstraction works on the design level.	Encapsulation works on the application level.
Abstraction is implemented to hide unnecessary data and withdrawing relevant data.	Encapsulation is the mechanism of hiding the code and the data together from the outside world or misuse.
It highlights what the work of an object instead of how the object works is	It focuses on the inner details of how the object works. Modifications can be done later to the settings.
Abstraction focuses on outside viewing, for example, shifting the car.	Encapsulation focuses on internal working or inner viewing, for example, the production of the car.
Abstraction is supported in Java with the interface and the abstract class.	Encapsulation is supported using, e.g. public, private and secure access modification systems.
In a nutshell, abstraction is hiding implementation with the help of an interface and an abstract class.	In a nutshell, encapsulation is hiding the data with the help of getters and setters.

NITIN MANGOTRA

https://www.educba.com/abstraction-vs-encapsulation/