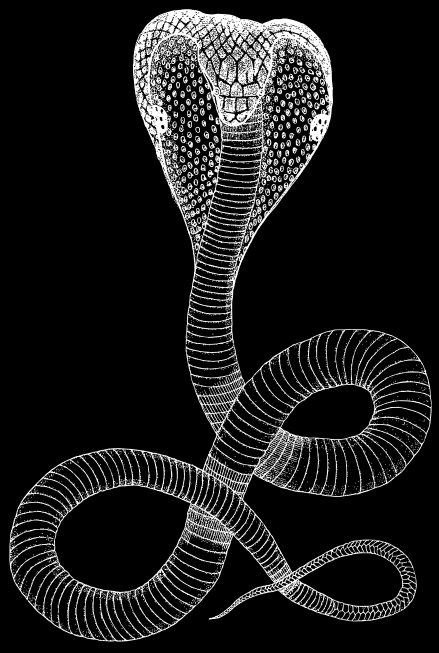


Crack Python Interviews

Part 1

Author
Arun Arunisto

About the Author

Arun Arunisto is a Python developer with over 3 years of experience in the industry. He is a computer enthusiast who learned programming on his own and has since become an expert in Python programming. Arun is passionate about teaching and has helped numerous students to learn Python programming.

Arun's vision is to simplify the learning process of Python programming for everyone. He believes that programming is for everyone and is committed to making programming accessible to all. His teaching style is unique, and he strives to make his classes fun and engaging.

Contents:

In his book, "How to Crack Python Interview," Arun Arunisto shares his insights and experience on how to prepare for a Python interview. The book covers a range of topics, including Python basics, data types, control structures, functions, object-oriented programming, and more. It also includes practical tips on how to ace the interview, such as common interview questions, tips for answering them, and strategies for demonstrating your knowledge and skills.

Whether you are a beginner or an experienced Python developer, this book will help you to prepare for your next Python interview and land your dream job.

Why Python?

Python is a popular programming language that is known for its simplicity, versatility, and readability. It is a high-level language that is easy to learn and has a vast ecosystem of libraries and frameworks that makes development faster and more efficient.

There are many reasons why Python has become one of the most popular programming languages. Here are some of the main reasons:

1. Simple and easy to learn: Python has a clean and straightforward syntax that is easy to read and understand. This makes it an excellent choice for beginners who are just starting with programming.

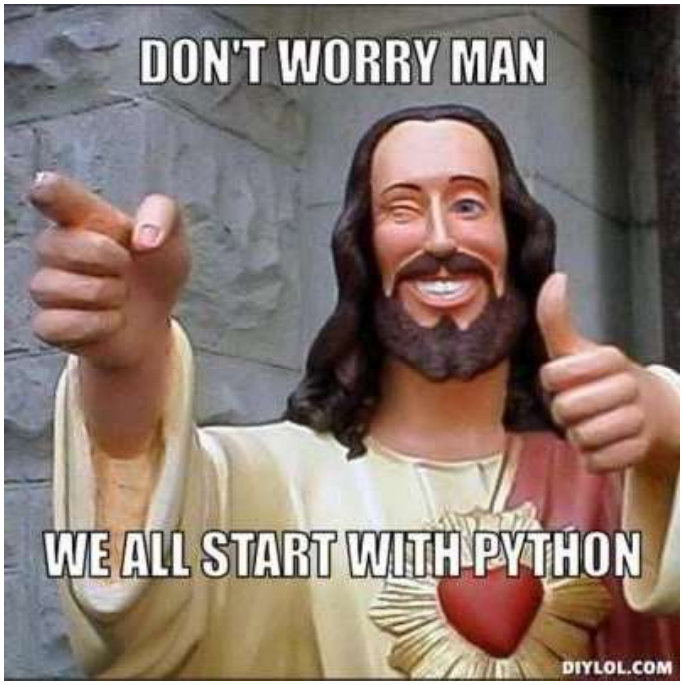
2. Large and active community: Python has a massive and active community of developers who contribute to various open-source projects and libraries. This means that there are a lot of resources available online, including documentation, tutorials, and forums.

3. Versatility: Python can be used for a wide range of applications, including web development, data analysis, artificial intelligence, scientific computing, game development, and more.

4. High-level language: Python is a high-level language, which means that it is closer to human language than machine language. This makes it easier to write and read code, and also reduces the amount of code needed to perform complex tasks.

5. Interpreted language: Python is an interpreted language, which means that there is no need for compilation or linking. This makes development faster and more efficient.

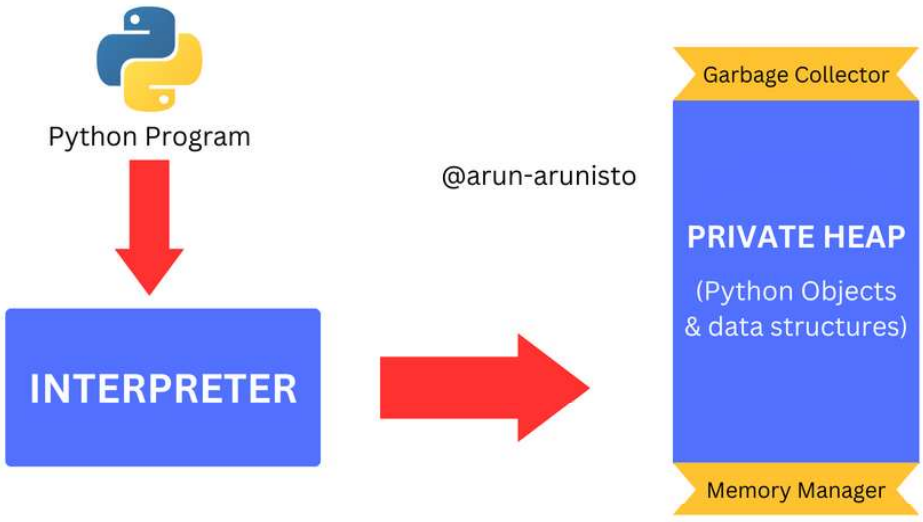
Overall, Python is a popular choice for programming due to its simplicity, versatility, and large community. It is widely used in various fields and has become one of the most in-demand programming languages in the job market.



Memory Management

Memory management in Python involves managing a private heap, which is a portion of memory that is exclusive to the Python process. This means that the operating system cannot allocate this memory to another process while it is being used by Python.

All Python objects and data structures, such as lists, dictionaries, and strings, are stored in the private heap. Python's memory manager automatically allocates and deallocates memory from this heap as needed, using techniques such as reference counting and garbage collection.



Memory Management in Python

.pyc files in python

In Python, .pyc files are compiled bytecode files that are generated by the Python interpreter when a .py file is imported or executed. When a .py file is imported or executed, the Python interpreter first checks to see if a corresponding .pyc file exists in the same directory. If the .pyc file exists and is up-to-date (i.e., the timestamp of the .pyc file is later than that of the .py file), then the interpreter uses the compiled bytecode in the .pyc file instead of recompiling the .py file.

The purpose of .pyc files is to improve the performance of Python programs by reducing the amount of time it takes to execute frequently used modules or scripts. By storing the compiled bytecode in a separate file, the Python interpreter can avoid the overhead of recompiling the .py file every time it is imported or executed.

Note that .pyc files are specific to a particular version of Python and platform, so they cannot be shared across different Python versions or platforms. Additionally, .pyc files do not contain any source code or comments, so they cannot be used for reverse engineering or code inspection purposes.

python is a interpreter language then why creating .pyc compiling files?

In Python, .pyc files are compiled bytecode files that are generated by the Python interpreter when a .py file is imported or executed. When a .py file is imported or executed, the Python interpreter first checks to see if a corresponding .pyc file exists in the same directory. If the .pyc file exists and is up-to-date (i.e., the timestamp of the .pyc file is later than that of the .py file), then the interpreter uses the compiled bytecode in the .pyc file instead of recompiling the .py file.

The purpose of .pyc files is to improve the performance of Python programs by reducing the amount of time it takes to execute frequently used modules or scripts. By storing the compiled bytecode in a separate file, the Python interpreter can avoid the overhead of recompiling the .py file every time it is imported or executed.

Note that .pyc files are specific to a particular version of Python and platform, so they cannot be shared across different Python versions or platforms. Additionally, .pyc files do not contain any source code or comments, so they cannot be used for reverse engineering or code inspection purposes.

What is PVM?

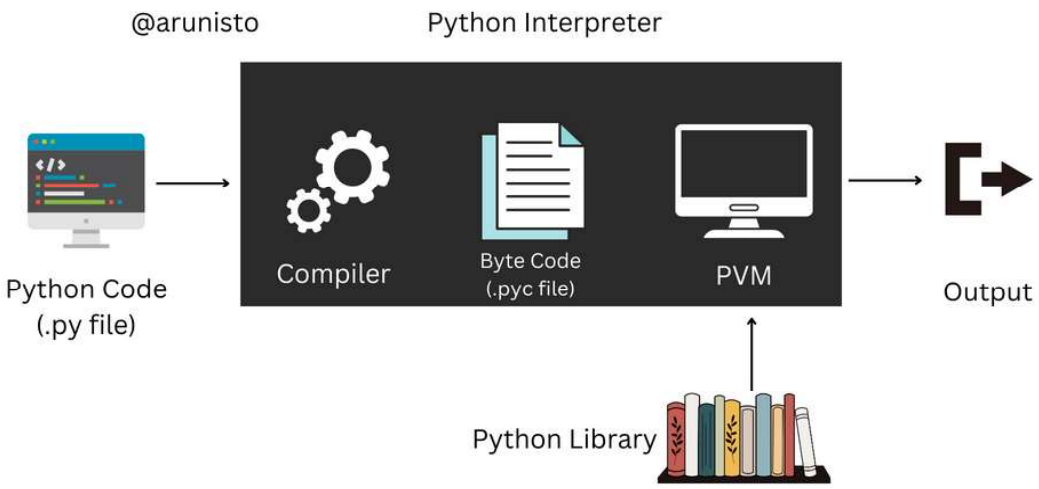
PVM stands for "Python Virtual Machine". It is a software layer that sits between the Python source code and the operating system. The PVM is responsible for interpreting Python bytecode and executing it on the computer's hardware.

When you run a Python program, the Python interpreter compiles the source code into bytecode, which is a lower-level representation of the code that can be executed by the PVM. The PVM then executes the bytecode on the computer's hardware, which produces the output of the program.

The PVM is designed to be portable and platform-independent, which means that Python programs can be run on any platform that has a PVM implementation. This is one of the strengths of Python as a programming language, as it allows developers to write code that can be run on a wide range of systems without needing to worry about the specifics of the underlying hardware or operating system.

Overall, the PVM is a critical component of the Python programming environment, as it is responsible for executing Python code and producing the output of Python programs.

How does Python works?



Python is an interpreted language, which means that the code is executed line by line at runtime by an interpreter. The interpreter reads the code, compiles it into bytecode, and then executes the bytecode.

Overall, Python works by compiling the source code into bytecode and then executing it in the PVM. The PVM manages the low-level tasks, such as memory allocation and garbage collection, allowing developers to focus on writing high-level code.

Pickling & Unpickling

Pickling and unpickling are techniques used in Python to serialize and deserialize Python objects. Serialization is the process of converting an object to a byte stream, and deserialization is the process of reconstructing the object from the byte stream.

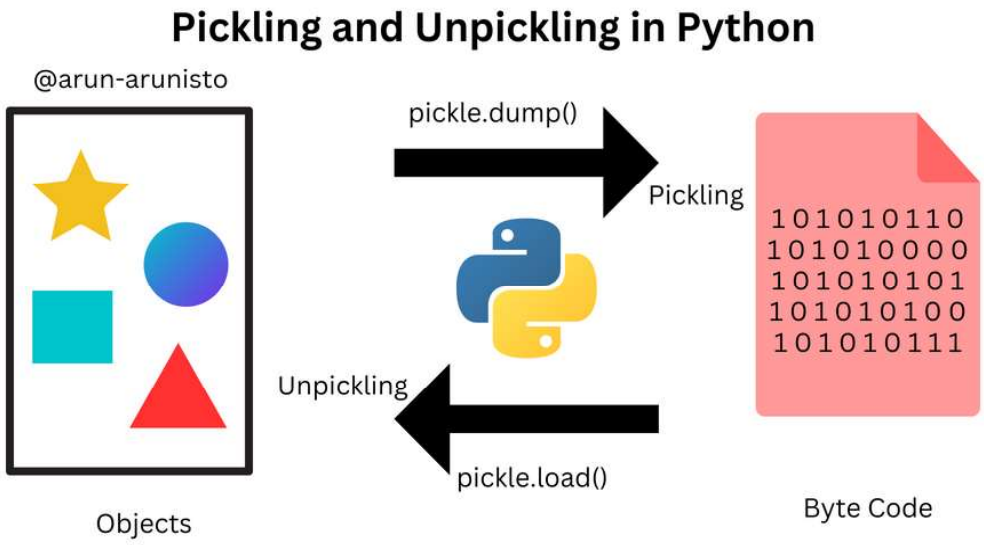
Pickling is the process of converting a Python object into a byte stream, which can be saved to a file or transmitted over a network. The pickle module provides the dump() function, which takes a Python object and a file object as arguments, and writes the serialized data to the file. The dumps() function can also be used to serialize an object into a byte string instead of a file.

Unpickling is the process of reconstructing a Python object from a byte stream. The pickle module provides the load() function, which takes a file object as an argument and reads the serialized data from the file, and returns the deserialized object. The loads() function can also be used to deserialize a byte string into an object.

The pickle module can serialize most Python objects, including classes, functions, and instances of custom classes.

However, there are some restrictions on what can be pickled, such as objects that have open file handles, network connections, or other external resources.

Pickling and unpickling can be useful in a variety of scenarios, such as caching, inter-process communication, and data exchange between different programming languages. However, it is important to be aware of the security implications of pickling and unpickling, as it can potentially execute arbitrary code if the data being unpickled is not trusted.



Sample Code

```
import pickle

my_object = {"Name":"arun arunisto", "Age":27,
"Job":"Developer"}

#pickle the object into a file
with open("my_object.pkl", "wb") as f:
    pickle.dump(my_object, f)

#pickling is the process converting an object into byte stream
#it's also called Serialization

#unpickilng is the process of reconstructing the object from
#the byte stram
#it's also called Deserialization

#unpickle the object from the file
with open("my_object.pkl", "rb") as f:
    loaded_object = pickle.load(f)

#printing the orginal and loaded data
print("Orginal data : ",my_object)
print("Loaded object : ",loaded_object)
```

List & Tuple

In Python, both lists and tuples are used to store collections of elements. However, there are some differences between them:

1. Mutability: Lists are mutable, which means you can add, remove, or modify elements in place. Tuples, on the other hand, are immutable, which means you cannot modify their contents once they are created.

2. Syntax: Lists are created using square brackets ('[]') and elements are separated by commas. Tuples are created using parentheses ('()') and elements are also separated by commas.

3. Usage: Lists are usually used for collections of elements that need to be modified or updated frequently, such as a list of items in a shopping cart. Tuples, on the other hand, are usually used for collections of elements that need to be accessed but not modified, such as a pair of latitude and longitude coordinates.

4. Performance: Tuples are generally more performant than lists, as they take up less memory and can be accessed more quickly. This is because tuples are immutable, so they don't require extra space to store information about resizing or allocation.

Here's an example to illustrate the differences between lists and tuples:

Sample Code

```
# Create a list of numbers
my_list = [1, 2, 3, 4, 5]

# Create a tuple of numbers
my_tuple = (1, 2, 3, 4, 5)

# Modify an element in the list
my_list[0] = 0

# This will raise a TypeError, as tuples are immutable
# my_tuple[0] = 0

# Print both the list and tuple
print('List:', my_list)
print('Tuple:', my_tuple)
```

Dictionary vs List

In Python, both lists and dictionaries are used to store collections of elements. However, there are some key differences between them:

1. Syntax: Lists are created using square brackets ('[]') and elements are separated by commas. Dictionaries are created using curly braces ('{}') and consist of key-value pairs separated by colons.

2. Accessing elements: Lists are accessed using an index, which is an integer value that represents the position of the element in the list. Dictionaries are accessed using a key, which is a unique identifier for a specific value.

3. Mutability: Lists are mutable, which means you can add, remove, or modify elements in place. Dictionaries are also mutable, which means you can add, remove, or modify key-value pairs.

4. Order: Lists maintain the order of their elements, which means the first element in the list will always be first, the second element will always be second, and so on. Dictionaries do not maintain any particular order, so the order in which you add key-value pairs may not be the order in which they are retrieved.

Here's an example to illustrate the differences between lists and dictionaries:

Sample Code :

```
#creating a new list
new_list = ["arun", "arunisto", "python", "19154Ever"]

#creating a new Dictionary
new_dict = {"name":"arun",
        "language":"python", "1915":"4EVER"}

#accessing elements
print(new_list[0]) #output : arun
print(new_dict["name"]) #output : arun

#modifying elements
new_list[0] = "arun arunisto"
new_dict["name"] = "arun arunisto"
```

for loop and pattern

A for loop is used to iterate over a sequence of elements, such as a list or a tuple, and execute a block of code for each element in the sequence. The number of iterations is predetermined by the length of the sequence.

Sample Code

```
#pyramid pattern
#odd numbers of "*"
space = 10 - 1
for i in range(1, 10):
   for j in range(space):
      print(end=" ")
   space-=1
   if i%2 != 0:
      for k in range(i):
         print("*", end=" ")
   print("\r")
```

fooBar program using For Loop

```
#fooBar Program
#divisible by 3 foo
#divisible by 5 bar
#divisible by 3 and 5 fooBar
for i in range(16):
    if i%3 == 0 and i%5 == 0:
        print("fooBar")
    elif i%3 == 0:
        print("foo")
    elif i%5 == 0:
        print("bar")
    else:
        print(i)
```

Note that for loop is an important topic on interviews it doesn't matter you're a fresher or experienced person. So, refer online resources for more programs related to this topic.

Recursion in Python

Recursion in Python is a process where a function calls itself repeatedly until a base condition is met. It is a powerful technique that can simplify complex problems by breaking them down into smaller sub-problems.

Recursion can be a useful tool for solving problems that involve repetitive or nested calculations. However, it can also be inefficient and consume a lot of memory if not used carefully. It's important to understand the limitations and trade-offs of recursion in order to use it effectively in your programs.

Sample Code

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * (factorial(n-1))

print(factorial(5))
```

Iterator in Python

Iterators allow you to iterate over a collection of elements one at a time, without having to load the entire collection into memory at once. This makes them useful for processing large data sets, where memory is limited.

In Python, an iterator is an object that implements the iterator protocol, which consists of two methods:

1. __iter__(): Returns the iterator object itself.
2. __next__(): Returns the next item in the sequence. Raises StopIteration when there are no more items to return.

Sample Code :

```
my_list = [1, 2, 3, 4, 5, 6]
my_iterator = iter(my_list)
print(next(my_iterator)) #1
print(next(my_iterator)) #2
print(next(my_iterator)) #3
print(next(my_iterator)) #4
print(next(my_iterator)) #5
print(next(my_iterator)) #6
print(next(my_iterator)) #StopIterationError
```

Generator in Python

In Python, a generator is a type of iterator that generates a sequence of values using the 'yield' keyword instead of returning a value with 'return'. Generators allow you to write iterators more quickly and easily than defining a class with '__iter__' and '__next__' methods.

Sample Code :

```
def square_numbers(n):
    for i in range(n):
        yield i**2

for square in square_numbers(5):
    print(square)
```

Decorator in Python

A decorator is a special type of function that can be used to modify the behavior of another function. Decorators are defined using the '@' symbol followed by the name of the decorator function, and are placed before the definition of the function they modify.

Decorators can be used for a wide variety of purposes, such as adding caching, logging, or authorization checks to functions, among other things.

Sample Code

```
def main_program(func):
    def hello():
        func()
        print("My age is : ",25)
    return hello

@main_program
def hi():
    print("My name is arun")

hi()
```

Multiple Inheritance

Multiple inheritance is a feature in Python that allows a class to inherit from more than one parent class. This means that a child class can have multiple base classes, and it inherits all the attributes and methods of each parent class.

Multiple inheritance can be a powerful tool for creating complex object hierarchies and can be used to compose classes in various ways to create new functionality. However, it can also lead to some issues, such as the diamond problem, which occurs when multiple base classes have a common ancestor class, leading to ambiguity in method resolution order.

```
#Sample Code
class Parent1:
    def hello():
        print("Hello")
class Parent2:
    def hi():
        print("Hi")
class Child(Parent1, Parent2):
    def main():
        Child.hi() #function of Parent2
        Child.hello() #function of Parent1
obj = Child
obj.main() #calling main function from child
```

Lambda functions

In Python, lambda is a keyword that is used to define small, anonymous functions. Anonymous functions are functions that are defined without a name. The lambda keyword is followed by the function arguments and a single expression that is evaluated and returned by the function.

Sample Code :

```
#define a lambda function that add two integers
add = lambda x, y: x+y

#printing the result
print(add(25, 36))
```

Python interview resources

- Python pattern programms : https://pynative.com/print-pattern-python-examples/

- Python interview questions : https://www.interviewbit.com/python-interview-questions/

Whether you are a beginner or an experienced programmer, this book is an excellent resource for preparing for a Python interview. By mastering the concepts covered in this book, you can increase your chances of cracking a Python interview and landing your dream job.

We hope that this book has been helpful to you in your Python interview preparation. Best of luck for your interview!
