PROGRAM CODE CLUB

# PYTHON BASIC INTERVIEW QUESTION

python

## Introduction of Python:

**Python** was developed by **Guido van Rossum** and was released first on February 20, 1991. It is one of the most widely used and loved programming languages and is interpreted in nature thereby providing flexibility of incorporating dynamic semantics. It is also a free and open-source language with very simple and clean syntax. Python also supports object-oriented programming and is most used to perform general-purpose programming.

## What is Python? What are the benefits of using Python:

Python is a high-level, interpreted, general-purpose programming language. Being a general-purpose language, it can be used to build almost any type of application with the right tools/libraries. Additionally, python supports objects, modules, threads, exception-handling, and automatic memory management which help in modelling real-world problems and building applications to solve these problems.

## Is Python a dynamically typed language?

Python is an interpreted language, executes each statement line by line and thus type-checking is done on the fly, during execution. Hence, Python is a Dynamically Typed Language.

## What are the benefits of using Python?

The benefits of using python are:

- Easy to use– Python is a high-level programming language that is easy to use, read, write, and learn.
- Interpreted language– Since python is interpreted language, it executes the code line by line and stops if an error occurs in any line.
- Dynamically typed– the developer does not assign data types to variables at the time of coding. It automatically gets assigned during execution.
- Free and open-source– Python is free to use and distribute. It is open source.
- Extensive support for libraries– Python has vast libraries that contain almost any function needed. It also further provides the facility to import other packages using Python Package Manager(pip).
- Portable– Python programs can run on any platform without requiring any change.
- The data structures used in python are user friendly.
- It provides more functionality with less coding.

## What is pep 8?

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

## What are Python namespaces?

A namespace in python refers to the name which is assigned to each object in python. The objects are variables and functions. As each object is created, its name along with space (the address of the outer function in which the object is), gets created. The namespaces are maintained in python like a dictionary where the key is the namespace and value are the address of the object. There 4 types of namespaces in python-

- **Built-in namespace**– These namespaces contain all the built-in objects in python and are available whenever python is running.
- **Global namespace**– These are namespaces for all the objects created at the level of the main program.
- **Enclosing namespaces**– These namespaces are at the higher level or outer function.
- **Local namespaces**– These namespaces are at the local or inner function.

## What are the common built-in data types in Python?

The common built-in data types in python are:

- **Numbers**– They include integers, floating-point numbers, and complex numbers. E.g., 1, 7.9,3+4i
- **List**– An ordered sequence of items is called a list. The elements of a list may belong to different data types. E.g. [5,'market',2.4]
- **Tuple**– It is also an ordered sequence of elements. Unlike lists, tuples are immutable, which means they can't be changed. E.g. (3,'tool',1)
- **String**– A sequence of characters is called a string. They are declared within single or double quotes. E.g., "Sana", 'She is going to the market', etc.
- **Set**– Sets are a collection of unique items that are not in order. E.g. {7,6,8}
- **Dictionary**– A dictionary stores values in key and value pairs where each value can be accessed through its key. The order of items is not important. E.g. {1:'apple',2:'mango}
- **Boolean**– There are 2 Boolean values- True and False.

## What are lists and tuples? What is the key difference between the two?

**Lists** and **Tuples** are both sequence data types that can store a collection of objects in Python. The objects stored in both sequences can have different data types. Lists are represented with **square brackets** ['Abhishek', 6, 0.19], while tuples are represented with **parentheses** ('Aditya', 5, 0.97).

The key difference between the two is that while lists are mutable, tuples on the other hand are **immutable** objects. This means that lists can be modified, appended, or sliced on the go but tuples remain constant and cannot be modified in any manner. You can run the following example on Python IDLE to confirm the difference:

```
my_tuple = ('Aditya', 6, 5, 0.97)
my_list = ['Abhishek', 6, 5, 0.97]
print(my_tuple[0])    # output => 'Aditya'
print(my_list[0])    # output => 'Abhishek'
my_tuple[0] = 'Adi'    # modifying tuple => throws an error
my_list[0] = 'Abhi'    # modifying list => list modified
print(my_tuple[0])    # output => 'Aditya'
print(my_list[0])    # output => 'Abhi'
```

## What is pass in Python?

The **pass** keyword represents a null operation in Python. It is generally used for the purpose of filling up empty blocks of code which may execute during runtime but has yet to be written. Without the pass statement in the following code, we may run into some errors during code execution.

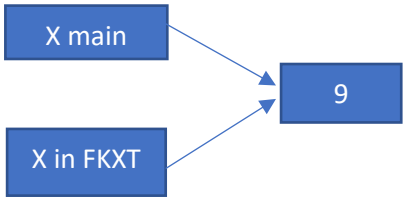
```
def passFucntion():
  # do nothing
  pass
passFucntion ()    # nothing happens
## Without the pass keyword
```

## What is Python's parameter passing mechanism?

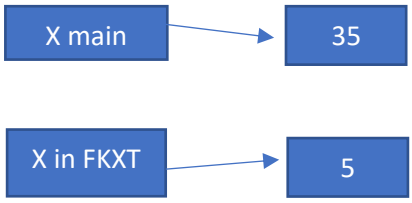
There are two parameters passing mechanism in Python:

- **Pass by references**
- **Pass by value.**

By default, all the parameters (arguments) are passed "by reference" to the functions. Thus, if you change the value of the parameter within a function, the change is reflected in the calling function as well. It indicates the original variable. **For example, if a variable is declared as a = 10, and passed to a function where it's value is modified to a = 20. Both the variables denote to the same value.**




The pass by value is that whenever we pass the arguments to the function only values pass to the function, no reference passes to the function. It makes it immutable that means not changeable. **Both variables hold the different values, and original value persists even after modifying in the function**




**Python's argument passing model is neither "Pass by Value" nor "Pass by Reference" but it is "Pass by Object Reference".**

## What are modules and packages in Python?

Python packages and Python modules are two mechanisms that allow for modular programming in Python. Modularizing has several advantages -

- **Simplicity**: Working on a single module helps you focus on a relatively small portion of the problem at hand. This makes development easier and less error prone.
- **Maintainability**: Modules are designed to enforce logical boundaries between different problem domains. If they are written in a manner that reduces interdependency, it is less likely that modifications in a module might impact other parts of the program.

- **Reusability**: Functions defined in a module can be easily reused by other parts of the application.
- **Scoping**: Modules typically define a separate namespace, which helps avoid confusion between identifiers from other parts of the program.

**Modules**, in general, are simply Python files with a **.py** extension and can have a set of functions, classes, or variables defined and implemented. They can be imported and initialized once using the import statement. If partial functionality is needed, import the requisite classes or functions using from foo import bar.

**Packages** allow for hierarchical structuring of the module namespace using dot notation. As, modules help avoid clashes between global variable names, in a similar manner, packages help avoid clashes between module names.

Creating a package is easy since it makes use of the system's inherent file structure. So just stuff the modules into a folder and there you have it, the folder name as the package name. Importing a module or its contents from this package requires the package name as prefix to the module name joined by a dot.

**Note: You can technically import the package as well, but alas, it doesn't import the modules within the package to the local namespace, thus, it is practically useless.**

## What are global, protected and private attributes in Python?

- **Global** variables are public variables that are defined in the global scope. To use the variable in the global scope inside a function, we use the `global` keyword.
- **Protected** attributes are attributes defined with an underscore prefixed to their identifier e.g., _aditya. They can still be accessed and modified from outside the class they are defined in, but a responsible developer should refrain from doing so.
- **Private** attributes are attributes with double underscore prefixed to their identifier e.g., __Abhishek. They cannot be accessed or modified from the outside directly and will result in an Attribute Error if such an attempt is made.

## What is the use of self in Python?

**Self** is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python. It binds the attributes with the given arguments. self is used in different places and often thought to be a keyword. But unlike in C++, self is not a keyword in Python.

## What is __init__ Or Constructor?

`__init__` is a constructor method in Python and is automatically called to allocate memory when a new object/instance is created. All classes have a **__init__** method associated with them. It helps in distinguishing methods and attributes of a class from local variables.

```
# class definition
class Student:
  def __init__(self, fname, lname, age, section):
      self.firstname = fname
      self.lastname = lname
      self.age = age
      self.section = section
# creating a new object
stu1 = Student("Dominic", "Avinash", 22, "A2")
```

## What is __del__ Or Destructors?

The `__del__` () method is a known as a **destructor** method in Python. It is called when all references to the object have been deleted i.e., when an object is garbage collected.

```
# Python program to illustrate destructor
class Employee:
   # Initializing
   def __init__(self):
      print ('Employee created.')
   # Deleting (Calling destructor)
   def __del__(self):
      print ('Destructor called, Employee deleted.')
```

## What is break, continue, and pass in Python?

| | |
|---|---|
| **Break** | The break statement terminates the loop immediately and the control flows to the statement after the body of the loop. |
| **Continue** | The continue statement terminates the current iteration of the statement, skips the rest of the code in the current iteration and the control flows to the next iteration of the loop. |
| **Pass** | As explained above, the pass keyword in Python |

```
pat = [1, 3, 2, 1, 2, 3, 1, 0, 1, 3]
for p in pat:
    pass
    if (p == 0):
        current = p
        break
    elif (p % 2 == 0):
        continue
    print(p)    # output => 1 3 1 3 1
print(current)    # output => 0
```

## What is slicing in Python?

A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g., `slice (start, end, step)`

```
a = ("a", "b", "c", "d", "e", "f", "g", "h")
x = slice(3, 5)
print(a[x]) // Output : ('d', 'e')
```

- Default value for **start** is 0, **stop** is number of items, **step** is 1.
- Slicing can be done on **strings, arrays, lists**, and **tuples**.

## What is pickling and unpickling?

**Pickling:**

- Pickling is the name of the serialization process in Python. Any object in Python can be serialized into a byte stream and dumped as a file in the memory. The process of pickling is compact, but pickle objects can be compressed further. Moreover, pickle keeps track of the objects it has serialized, and the serialization is portable across versions.
- The function used for the above process is `pickle.dump()`.

**Unpickling:**

- Unpickling is the complete inverse of pickling. It deserializes the byte stream to recreate the objects stored in the file and loads the object to memory.
- The function used for the above process is `pickle.load()`.

**Note:** *Python has another, more primitive, serialization module called **Marshall**, which exists primarily to **support .pyc files** in Python and **differs significantly from the pickle**.*

## What are generators in Python?

Generators are functions that return an iterable collection of items, one at a time, in a set manner. Generators, in general, are used to create iterators with a different approach. They employ the use of `yield` keyword rather than `return` to return a **generator** object.

## What are iterators in Python?

An iterator is an object. `__iter__()` method initializes an iterator. It has a `__next__()` method which returns the next item in iteration and points to the next element. Upon reaching the end of iterable object __next__() must return Stop Iteration exception. It is also self-iterable. Iterators are objects with which we can iterate over iterable objects like lists, strings, etc.

## What is lambda in Python? Why is it used?

Lambda is an anonymous function in Python, that can accept any number of arguments, but can only have a single expression. It is generally used in situations requiring an anonymous function for a short time period. Lambda functions can be used in either of the two ways:

- Assigning lambda functions to a variable:
- Wrapping lambda functions inside another function:

```
Assigning into Lambada function to a variable
mule = lambda a, b: a * b
print (mule (2, 5)) // output => 10

Wrapping lambda functions inside another function:
def myWrapper(n):
 return lambda a: a * n
multiverb = myWrapper(5)
print(multiverb(2))    # output => 10
```

## What does *args and **kwargs mean?

### *args

- *args is a special syntax used in the function definition to pass variable-length arguments.
- "*" means variable length and "args" is the name used by convention. You can use any other.

### **kwargs

- **kwargs is a special syntax used in the function definition to pass variable-length keyworded arguments.

- Here, also, “kwargs” is used just by convention. You can use any other name.
- Keyworded argument means a variable that has a name when passed to a function.
- It is a dictionary of the variable names and its value.

## What is the difference between xrange and range in Python?

**xrange()** and **range()** are quite similar in terms of functionality. They both generate a sequence of integers, with the only difference that `range()` returns a **Python list**, whereas, `xrange()` returns an **xrange object**.

**So how does that make a difference?** It sure does, because unlike range(), xrange() doesn't generate a static list, it creates the value on the go. This technique is commonly used with an object-type **generator** and has been termed as "**yielding**".

**Yielding** is crucial in applications where memory is a constraint. Creating a static list as in range() can lead to a `Memory Error` in such conditions, while, xrange() can handle it optimally by using just enough memory for the generator (significantly less in comparison).

**Note**: **xrange** has been **deprecated** as of **Python 3.x**. Now `range` does exactly the same as what `xrange` used to do in **Python 2.x**, since it was way better to use xrange() than the original range() function in Python 2.x.

## Explain split() and join() functions in Python?

**split()** function to split a string based on a delimiter to a list of strings.

**join()** function to join a list of strings based on a delimiter to give a single string.

```
string = "This is a string."
string_list = string.split(' ') #delimiter is ‘space’ character or ‘ ‘
```

```
print(string_list) #output: ['This', 'is', 'a', 'string.']
print(' '.join(string_list)) #output: This is a string.
```

## What are negative indexes and why are they used?

- Negative indexes are the indexes from the end of the list or tuple or string.
- **Arr[-1]** means the last element of array **Arr[]**

```
arr = [1, 2, 3, 4, 5, 6] #get the last element
print(arr[-1]) #output 6#get the second last element
print(arr[-2]) #output 5
```

## What are decorators in Python?

**Decorators** in Python are essentially functions that add functionality to an existing function in Python without changing the structure of the function itself. They are represented the `@decorator_name` in Python and are called in a bottom-up fashion.

```
# decorator function to convert to lowercase
def lowercase_decorator(function):
   def wrapper():
       func = function()
       string_lowercase = func.lower()
       return string_lowercase
   return wrapper
# decorator function to split words
def splitter_decorator(function):
   def wrapper():
       func = function()
       string_split = func.split()
       return string_split
   return wrapper
@splitter_decorator # this is executed next
@lowercase_decorator # this is executed first
```

## What are Dict and List comprehensions?

Python comprehensions, like decorators, are **syntactic sugar** constructs that help **build altered** and **filtered lists**, dictionaries, or sets from a given list, dictionary, or set. Using comprehensions saves a lot of time and code that might be considerably more verbose (containing more lines of code

- **Performing mathematical operations on the entire list**

```
my_list = [2, 3, 5, 7, 11]
squared_list = [x**2 for x in my_list]    # list comprehension
# output => [4 , 9 , 25 , 49 , 121]
squared_dict = {x:x**2 for x in my_list}    # dict comprehension
# output => {11: 121, 2: 4 , 3: 9 , 5: 25 , 7: 49}
```

- **Performing conditional filtering operations on the entire list**

```
my_list = [2, 3, 5, 7, 11]
squared_list = [x**2 for x in my_list if x%2 != 0]    # list comprehension
# output => [9 , 25 , 49 , 121]
squared_dict = {x:x**2 for x in my_list if x%2 != 0}    # dict
comprehension
# output => {11: 121, 3: 9 , 5: 25 , 7: 49}
```

- **Combining multiple lists into one**

Comprehensions allow for multiple iterators and hence, can be used to combine multiple lists into one.

```
a = [1, 2, 3]
b = [7, 8, 9]
[(x + y) for (x,y) in zip(a,b)]  # parallel iterators
# output => [8, 10, 12]
[(x,y) for x in a for y in b]    # nested iterators
# output => [(1, 7), (1, 8), (1, 9), (2, 7), (2, 8), (2, 9), (3, 7), (3,
8), (3, 9)]
```

- **Flattening a multi-dimensional list**

A similar approach of nested iterators (as above) can be applied to flatten a multi-dimensional list or work upon its inner elements.

```
my_list = [[10,20,30],[40,50,60],[70,80,90]]
flattened = [x for temp in my_list for x in temp]
# output => [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

**Note:** List comprehensions have the same effect as the map method in other languages. They follow the mathematical set builder notation rather than map and filter functions in Python.

## What are modules and packages in Python?

Python packages and Python modules are two mechanisms that allow for **modular programming** in Python. Modularizing has several advantages -

- **Simplicity**: Working on a single module helps you focus on a relatively small portion of the problem at hand. This makes development easier and less error prone.
- **Maintainability**: Modules are designed to enforce logical boundaries between different problem domains. If they are written in a manner that reduces interdependency, it is less likely that modifications in a module might impact other parts of the program.

- **Reusability**: Functions defined in a module can be easily reused by other parts of the application.
- **Scoping**: Modules typically define a separate namespace, which helps avoid confusion between identifiers from other parts of the program.

**Modules**, in general, are simply Python files with a .py extension and can have a set of functions, classes, or variables defined and implemented. They can be imported and initialized once using the `import` statement. If partial functionality is needed, import the requisite classes or functions using `from foo import bar`.

**Packages** allow for hierarchial structuring of the module namespace using **dot notation**. As, **modules** help avoid clashes between global variable names, in a similar manner, **packages** help avoid clashes between module names.
Creating a package is easy since it makes use of the system's inherent file structure. So just stuff the modules into a folder and there you have it, the folder name as the package name. Importing a module or its contents from this package requires the package name as prefix to the module name joined by a dot.

***Note: You can technically import the package as well, but alas, it doesn't import the modules within the package to the local namespace, thus, it is practically useless.***

## How is memory managed in Python?

- Memory management in Python is handled by the **Python Memory Manager**. The memory allocated by the manager is in form of a **private heap space** dedicated to Python. All Python objects are stored in this heap and being private, it is inaccessible to the programmer. Though, python does provide some core API functions to work upon the private heap space.
- Additionally, Python has an in-built garbage collection to recycle the unused memory for the private heap space.