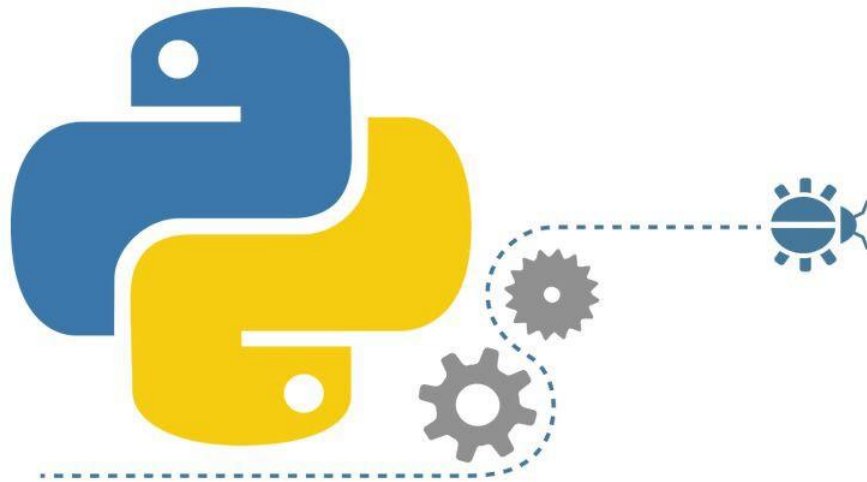# Python Tutorial Android Application (Published)

- Offline App, Internet required only for the quiz module.
- Quiz module: 100+ multiple choice interview level questions.
- Basic and advance Python programming units.
- Data Structures in Python with implementation code.
- Algorithms and their implementation in Python.
- Rich UI design.
- Easy navigation between different sections.
- Eye friendly reading interface.
- Light - No background services, No annoying notifications.

Content -
Basic
Installation
Variables
Data types
Conditional Statement
Loops
Control Statement
Functions, etc.

Advanced
Conversion of data types
Using else with loops
Anonymous Functions
Classes
File operations
Managing of file and directories in Python
Modules
Errors and Exception, etc.

Data Structures
Lists
Sets

Tuples
Dictionary, and more

Algorithms
Searching, Sorting, Shortest path, etc.

Actual application screenshots below -

Conversion of data types

Using Else with loops

Anonymous Functions

Different argument types for Functions

Python classes

Python classes - Extended

Inheritance in Python classes

Python Files open/close

Python Files read/write

Using Python   and
directories   Advanced

Python modules

Python,Data Structures and Algori...

What is the output of the following?
print('{0:.2}'.format(1/3))

○ Error

◉ 0.333333   CHECK

○ 0.33

Status: Incorrect!

Answer: 0.33

Score: 0 / 1

EXIT   NEXT ->

Python,Data Structures and Algori...

What is the output of the following?
print('ab,12'.isalnum())

○ None

○ True   CHECK

◉ False

Status: Correct!

Answer: False

Score: 2 / 3

EXIT   NEXT ->

The If condition

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Conditional statements give us this ability. The simplest form is the if statement, which has the genaral form:

if BOOLEAN EXPRESSION:
  STATEMENTS

A few important things to note about if statements:

·The colon (:) is significant and required. It separates the header of the compound statement from the body.

·The line after the colon must be indented. It is standard in Python to use four spaces for indenting.

·All lines indented the same amount after the colon will be executed whenever the BOOLEAN_EXPRESSION is true.

Here is an example:

```
food = 'spam'
if food == 'spam':
    print('Ummmm, my favorite!')
    print('I feel like saving it 100 times...')
```

Opening and CLosing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files. Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it. Syntax :

```
file object = open(file_name [, access_mode][,
buffering])
```

Here are parameter details :

·**file_name** – The file_name argument is a string value that contains the name of the file that you want to access.

·**access_mode** – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter

---

Implementing queue in Python

It is again appropriate to create a new class for the implementation of the abstract data type queue. As before, we will use the power and simplicity of the list collection to build the internal representation of the queue. We need to decide which end of the list to use as the rear and which to use as the front. The implementation shown below assumes that the rear is at position 0 in the list. This allows us to use the **insert** function on lists to add new elements to the rear of the queue. The **pop** operation can be used to remove the front element (the last element of the list).

```
class Queue:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.insert(0,item)
    def dequeue(self):
        return self.items.pop()
    def size(self):
        return len(self.items)
q=Queue()
q.enqueue(4)
q.enqueue('dog')
q.enqueue(True)
print(q.size())
```

Quick Sort

```python
# Python program for implementation of Quicksort
Sort

# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot
def partition(arr,low,high):
    i = ( low-1 ) # index of smaller element
    pivot = arr[high] # pivot

    for j in range(low , high):

        # If current element is smaller than or
        # equal to pivot
        if arr[j]
            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index
```

Dijkstra's shortest path algorithm

```python
# Python program for Dijkstra's single
# source shortest path algorithm. The program is
# for adjacency matrix representation of the graph

# Library for INT_MAX
import sys

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
            for row in range(vertices)]

    def printSolution(self, dist):
        print Vertex tDistance from Source
        for node in range(self.V):
            print node,t,dist[node]

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minDistance(self, dist, sptSet):

        # Initilaize minimum distance for next node
        min = sys.maxint

        # Search not nearest vertex not in the
```

Menu

Basics

Advanced features

Data Structures

Algorithms