**INTRODUCTION TO JAVA**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales.[[26]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-7-26) Van Rossum led the language community until stepping down as leader in July 2018.[[27]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-27)[[28]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-28)

Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming). It also has a comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).[[29]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-About-29)

Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython" \o "CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open-source_software)software[[30]](https://en.wikipedia.org/wiki/Python_(programming_language)" \l "cite_note-30) and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation).

Python was conceived in the late 1980s[[31]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-venners-interview-pt-1-31) by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) (CWI) in the [Netherlands](https://en.wikipedia.org/wiki/Netherlands) as a successor to the [ABC language](https://en.wikipedia.org/wiki/ABC_(programming_language)) (itself inspired by [SETL](https://en.wikipedia.org/wiki/SETL))[[32]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-12-32), capable of [exception handling](https://en.wikipedia.org/wiki/Exception_handling) and interfacing with the [Amoeba](https://en.wikipedia.org/wiki/Amoeba_(operating_system)) operating system.[[7]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-faq-created-7) Its implementation began in December 1989.[[33]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-timeline-of-python-33) Van Rossum's long influence on Python is reflected in the title given to him by the Python community: [*Benevolent Dictator For Life*](https://en.wikipedia.org/wiki/Benevolent_Dictator_For_Life) (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.[[34]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-lj-bdfl-resignation-34)

Python 2.0 was released on 16 October 2000 with many major new features, including a [cycle-detecting](https://en.wikipedia.org/wiki/Cycle_detection) [garbage collector](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) and support for [Unicode](https://en.wikipedia.org/wiki/Unicode).[[35]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-newin-2.0-35)

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility).[[36]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-3.0-release-36) Many of its major features were [backported](https://en.wikipedia.org/wiki/Backporting" \o "Backporting) to Python 2.6.x[[37]](https://en.wikipedia.org/wiki/Python_(programming_language)" \l "cite_note-pep-3000-37) and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.[[38]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-38)

Python 2.7's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)) date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[[39]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-39)[[40]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-40) In January 2017, Google announced work on a Python 2.7 to [Go](https://en.wikipedia.org/wiki/Go_(programming_language)) [transcompiler](https://en.wikipedia.org/wiki/Transcompiler" \o "Transcompiler) to improve performance under concurrent workloads.[[41]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-41)

Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of its features support [functional programming](https://en.wikipedia.org/wiki/Functional_programming) and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including by [metaprogramming](https://en.wikipedia.org/wiki/Metaprogramming" \o "Metaprogramming)[[42]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-13-42) and [metaobjects](https://en.wikipedia.org/wiki/Metaobject" \o "Metaobject) (magic methods)).[[43]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-14-43) Many other paradigms are supported via extensions, including [design by contract](https://en.wikipedia.org/wiki/Design_by_contract)[[44]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-15-44)[[45]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-16-45) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming).[[46]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-17-46)

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing), and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages))([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

Python's design offers some support for [functional programming](https://en.wikipedia.org/wiki/Functional_programming) in the [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)) tradition. It has filter(), map(), and reduce() functions; [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension), [dictionaries](https://en.wikipedia.org/wiki/Associative_array), sets and [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expressions.[[47]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-59-47) The standard library has two modules (itertools and functools) that implement functional tools borrowed from [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).[[48]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-18-48)

The language's core philosophy is summarized in the document *The*[*Zen of Python*](https://en.wikipedia.org/wiki/Zen_of_Python) (*PEP 20*), which includes [aphorisms](https://en.wikipedia.org/wiki/Aphorism) such as:[[49]](https://en.wikipedia.org/wiki/Python_(programming_language)" \l "cite_note-PEP20-49)

* Beautiful is better than ugly
* Explicit is better than implicit
* Simple is better than complex
* Complex is better than complicated
* Readability counts

Explanation about Mini Project

Simple Hotel Management System project is written in Python. The project file contains a python script (Hotelms.py). This is a simple console based system which is very easy to understand and use. Talking about the system, it contains all the basic functions which include entering customer’s data, calculating room rent, restaurant bill, laundry bill, game bill, and total cost. In this mini project, there is no such login system. This means he/she can use all those available features easily without any restriction. It is too easy to use, he/she can check the total cost of staying in the hotel easily with each and every detail.

Talking about the features of this Simple Hotel Management System, at first, the user has to enter his/her data. It includes the name of the user, address, check-in, and check-out dates. The user can calculate room rents. Inside this section, there are total four types of room with different prices. After selecting the room type, the system asks to enter the number of nights spent in order to calculate room rent. This simple system also contains other functions such as calculating restaurant, laundry and game bill. When the user selects to calculate restaurant bill, the system displays a small menu. From there the user has to select foods and then it displays the total restaurant bill. The other remaining features; calculating laundry and game bill also follows the same procedure as of calculating restaurant bill.

At last, after all these calculations the user can know about their total cost of staying easily. In this feature, the system provides his/her details, with the room number, room rent, food, laundry and games bill. The total sum is displayed to the users with some additional charges. This simple console based Hotel Management system provides the simplest management of hotel service and transaction. In short, this projects mainly focus on adding and calculating results. There’s no external database connection file used in this mini project to save user’s data permanently.

**Features:**

1. Enter Customer’s detail
2. Calculate Food, Laundry and Game Bills
3. Calculate Room Rent
4. Total Bill

List of Python Concepts used in Mini Project

1. LIBARIES

2. FUNCTIONS

3. I/O STATEMENTS

4. CONTROL STRUCTURES

5. CLASS

6. OPERATORS

7. ARRAYS

Explanation of Each & Every Python Concept used in Mini Project

1] LIBARIES

Python's large [standard library](https://en.wikipedia.org/wiki/Standard_library), commonly cited as one of its greatest strengths,[[94]](https://en.wikipedia.org/wiki/Python_(programming_language)" \l "cite_note-AutoNT-86-94) provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as [MIME](https://en.wikipedia.org/wiki/MIME) and [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) are supported. It includes modules for creating [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface), connecting to [relational databases](https://en.wikipedia.org/wiki/Relational_database), [generating pseudorandom numbers](https://en.wikipedia.org/wiki/Pseudorandom_number_generator), arithmetic with arbitrary precision decimals,[[95]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-88-95) manipulating [regular expressions](https://en.wikipedia.org/wiki/Regular_expression), and [unit testing](https://en.wikipedia.org/wiki/Unit_testing).

Some parts of the standard library are covered by specifications (for example, the [Web Server Gateway Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) (WSGI) implementation wsgiref follows PEP 333[[96]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-89-96)), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the [Python Package Index](https://en.wikipedia.org/wiki/Python_Package_Index) (PyPI), the official repository for third-party Python software, contains over 130,000[[97]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-97) packages with a wide range of functionality, including:

* Graphical user interfaces
* Web frameworks
* Multimedia
* Databases
* Networking
* Test frameworks
* Automation
* Web scraping[[98]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-98)
* Documentation
* System administration
* Scientific computing
* Text processing
* Image processing

2] FUNCTIONS

SYNTAX

Functions are a construct to programs. They are known in most programming languages, sometimes also called subroutines or procedures. Functions are used to utilize code in more than one place in a program. The only way without functions to reuse code consists in copying the code.   
  
A function in Python is defined by a def statement. The general syntax looks like this:

def function-name(Parameter list):

statements, i.e. the function body

The parameter list consists of none or more parameters. Parameters are called arguments, if the function is called. The function body consists of indented statements. The function body gets executed every time the function is called.   
Parameter can be mandatory or optional. The optional parameters (zero or more) must follow the mandatory parameters.   
  
Function bodies can contain a return statement. It can be anywhere in the function body. This statement ends the execution of the function call and "returns" the result, i.e. the value of the expression following the return keyword, to the caller. If there is no return statement in the function code, the function ends, when the control flow reaches the end of the function body.   
Example:

>>> def add(x, y):

... """Return x plus y"""

... return x + y

...

>>>

In the following interactive session, the function we previously defined will be called:

>>> add(4,5)

9

>>> add(8,3)

11

>>>

3] DATA TYPES AND VARIABLES

Even if you think that you know a lot about data types and variables, because you have programmed lower level languages like C, C++ or other similar programming languages, we would recommend to read this chapter. Data types and variables in Python are different in some aspects from other programming languages. There are integers, floating point numbers, strings, and many more, but things are not the same as in C or C++. If you want to use lists in C e.g., you will have to construe the data type list from scratch, i.e. design memory structure and the allocation management. You will have to implement the necessary search and access methods as well. Python provides power data types like lists as a genuine part of the language.

VARIABLES

As the name implies, a variable is something which can change. A variable is a way of referring to a memory location used by a computer program. A variable is a symbolic name for this physical location. This memory location contains values, like numbers, text or more complicated types.   
A variable can be seen as a container (or some say a pigeonhole) to store certain values. While the program is running, variables are accessed and sometimes changed, i.e. a new value will be assigned to the variable.   
One of the main differences between Python and strongly-typed languages like C, C++ or Java is the way it deals with types. In strongly-typed languages every variable must have a unique data type. E.g. if a variable is of type integer, solely integers can be saved in the variable. In Java or C, every variable has to be declared before it can be used. Declaring a variable means binding it to a data type.   
Declaration of variables is not required in Python. If there is need of a variable, you think of a name and start using it as a variable.   
Another remarkable aspect of Python: Not only the value of a variable may change during program execution but the type as well. You can assign an integer value to a variable, use it as an integer for a while and then assign a string to the variable.   
In the following line of code, we assign the value 42 to a variable:

i = 42

The equal "=" sign in the assignment shouldn't be seen as "is equal to". It should be "read" or interpreted as "is set to", meaning in our example "the variable **i** is set to 42". Now we will increase the value of this variable by 1: 

>>> i = i + 1

>>> print i

43

>>>

Coding/Implementation

from collections import namedtuple

import datetime

Reservation = namedtuple('Reservation','room arr\_date dept\_date guest\_name confirmation\_num')

#----------global variables / lists----------#

confirmation\_counter = 0

bedroom\_list = []

reservation\_list = []

#main function

def Anteater\_BandB (file\_name:str)-> None:

'''main function. reads a file named file\_name'''

infile = open(file\_name, 'r')

data = infile.readlines()

infile.close()

for line in data:

line = line.strip()

line\_reader(line)

def line\_reader(l: str) -> None:

'''takes in one line of input and calls appropriate functions'''

command = l[:2].upper()

rest\_of\_input = l[2:].strip()

if command == '\*\*':

pass;

elif command == 'AB':

add\_bedroom(rest\_of\_input)

elif command == 'BL':

display\_bedroom\_list()

elif command == 'PL':

print\_line(rest\_of\_input)

elif command == 'BD':

delete\_bedroom(rest\_of\_input)

elif command == 'NR':

new\_reservation(rest\_of\_input)

elif command == 'RL':

display\_reservation\_list()

elif command == 'RD':

delete\_reservation(rest\_of\_input)

elif command == 'RB':

reservations\_by\_bedroom(rest\_of\_input)

elif command == 'RC':

reservations\_by\_guest(rest\_of\_input)

elif command == 'LA':

list\_arrivals(rest\_of\_input)

elif command == 'LD':

list\_departures(rest\_of\_input)

elif command =='LF':

list\_free\_beds(rest\_of\_input)

elif command == 'LO':

list\_occupied(rest\_of\_input)

def dashes():

return '------------------------------------'

#AB

def add\_bedroom(room: str)-> None:

'''takes in a room number as a string and puts it into bedroom\_list'''

global bedroom\_list

if room in bedroom\_list:

print('The bedroom is already in the list.')

else:

bedroom\_list.append(room)

#BL

def display\_bedroom\_list():

'''prints items in bedroom\_list'''

global bedroom\_list

print('Number of bedrooms in service:\t', len(bedroom\_list))

print(dashes())

for bed in bedroom\_list:

print(bed)

#PL

def print\_line(r: str):

print(r)

#BD

def delete\_bedroom(room:str)-> None:

'''deletes specified room from the list. print error message if

room isn't on the list'''

global bedroom\_list

if room in bedroom\_list:

bedroom\_list.remove(room)

cancel\_room\_reservations(room)

else:

print('Sorry, can\'t delete room '+room+'; it is not in service now')

def compare\_date(date1:str,date2:str)->bool:

'''compares two dates. returns true if date 2 is after date 1'''

date1 = date1.split('/')

date\_1 = datetime.date(int(date1[2]),int(date1[0]),int(date1[1]))

date2 = date2.split('/')

date\_2 = datetime.date(int(date2[2]),int(date2[0]),int(date2[1]))

if date\_2>date\_1:

return True

else:

return False

def date(date:str)->int:

date = date.split('/')

result = datetime.date(int(date[2]),int(date[0]),int(date[1]))

return result

def conf\_num(r: Reservation) -> int:

return r.confirmation\_num

#NR

def new\_reservation(rest\_input: str)->None:

'''creates a new reservation namedtuple and adds it to reservation\_list'''

global reservation\_list

global bedroom\_list

global confirmation\_counter

#chop up input to get variables

parts = rest\_input.split()

room\_request = parts[0]

arrival = parts[1]

departure = parts[2]

name = ''

for item in parts[3:]:

name+=item

name+=' '

if (room\_request in bedroom\_list) and (allow\_reservation(arrival, departure)) and room\_not\_taken(room\_request) :

confirmation\_counter+=1

reservation = Reservation(room\_request, arrival, departure, name, confirmation\_counter)

reservation\_list.append(reservation)

print('Reserving room '+room\_request+' for '+name+' -- Confirmation # ' + str(confirmation\_counter))

print('(arriving ' + arrival + ', departing ' + departure + ' )')

elif allow\_reservation(arrival, departure) == False:

print("Sorry, can't reserve room ",room\_request,'(',arrival,' to ',departure,"); \n can't leave before you arrive.")

elif room\_not\_taken(room\_request)==False:

print("Sorry, can't reserve room ",room\_request,'(',arrival,' to ',departure,"); \n it is already booked (conf # ",str(confirmation\_counter))

else:

print("Sorry, can't reserve room", room\_request,'; room not in service')

#RL

def display\_reservation\_list():

global reservation\_list

print('Number of reservations:\t' + str(len(reservation\_list)))

print('{:>3}{:>4}{:>11}{:>11}{}{}'.format('No.','Rm.','Arrive','Depart',' ','Guest'))

print(dashes())

for r in reservation\_list:

print('{:>3}{:>4}{:>11}{:>11}{}{}'.format(

str(r.confirmation\_num),r.room,r.arr\_date,r.dept\_date,' ',r.guest\_name))

#RD

def delete\_reservation(num: str):

'''takes in a confirmation number and deletes the reservation with that confirmation number'''

global reservation\_list

#reservation\_list.sort(key = conf\_num, reverse=False)

confirmation\_list = []

for r in reservation\_list:

confirmation\_list.append(r.confirmation\_num)

if (int(num) in confirmation\_list):

reservation\_index = confirmation\_list.index(int(num)) #find where the reservation is in the list

reservation\_list.remove(reservation\_list[reservation\_index]) #delete the reservation with that index

else:

print("Sorry, can't cancel reservation; no confirmation number " + num)

#First: reject if arrival of guest A is later than departure date of guest A

def allow\_reservation(arr: str, dept: str)-> bool:

'''takes in two dates as strings, converts them to dates to compare them,

and determines whether, based on the arr and dept date, the reservation is valid

'''

if date(arr)>=date(dept):

#print('can\'t leave before you arrive')

return False

return True

#Second: check conflicts with existing reservations

def room\_not\_taken(room\_req: str) -> bool:

'''return true if room is taken'''

#based on whether bedroom is free

global reservation\_list

reserved\_rooms = []

for r in reservation\_list:

reserved\_rooms.append(r.room) #list of strings of taken rooms

if (room\_req not in reserved\_rooms):

return True

return False

def reservations\_conflict(r1:Reservation, r2:Reservation)-> bool:

'''takes two reservations and compares them. return true if they conflict'''

if (date(r1.arr\_date)>=date(r2.arr\_date) and date(r1.arr\_date)<date(r2.dept\_date)) or (date(r1.dept\_date)>=date(r2.arr\_date) and date(r1.dept\_date)<date(r2.dept\_date)):

print('Sorry, can\'t reserve room '+room\_request+'\t('+arrival+' to '+departure+');')

print('it\'s already been booked')

return True

return False

#Finally, if user deletes bedroom, all reservations for that room are cancelled

def cancel\_room\_reservations(room:str):

for r in reservation\_list:

if r.room == room:

print('Deleting room',room,'forces cancellation of this reservation:')

print('\t',r.guest\_name,'arriving',r.arr\_date,'and departing',r.dept\_date,'(Conf. #',r.confirmation\_num,')')

delete\_reservation(r.confirmation\_num)

#RB

def reservations\_by\_bedroom(line:str):

global reservation\_list

bedroom\_reserve\_list= []

bedroom\_num = line

for r in reservation\_list:

if r.room == bedroom\_num:

bedroom\_reserve\_list.append(r)

print("Reservations for room " + line +':')

for re in bedroom\_reserve\_list:

print(re.arr\_date,' to ',re.dept\_date, re.guest\_name)

#RC

def reservations\_by\_guest(line:str):

guest\_reserve\_list = []

guest\_name = line

print('Reservation for',guest\_name)

for r in reservation\_list:

if r.guest\_name == guest\_name:

print(r.arr\_date + ' to ' + r.dept\_date + ': room ' + r.room)

def display\_guest(rl:list) -> None:

'''takes in a reservation list and prints out guest name as well as room number

'''

for r in rl:

print(r.guest\_name+ '(room '+ r.room + ')')

def reserved\_rooms(rl: list) -> list:

'''takes in a list of reservations and returns a list of reserved rooms'''

reserved = []

for r in rl:

reserved.append(str(r.room))

return reserved

#LA

def list\_arrivals(line:str):

guest\_arrival\_list = []

guest\_arrival = date(line)

for r in reservation\_list:

if guest\_arrival == date(r.arr\_date):

guest\_arrival\_list.append(r)

print('Guests arriving on '+ line+ ':')

display\_guest(guest\_arrival\_list)

#LD

def list\_departures(line:str):

guest\_departure\_list = []

guest\_departure = date(line)

for r in reservation\_list:

if guest\_departure == date(r.dept\_date):

guest\_departure\_list.append(r)

print('Guests departing on '+ line+ ':')

display\_guest(guest\_departure\_list)

#LF

def list\_free\_beds(line:str):

global bedroom\_list

bedroom\_requests = []

two\_dates = line.split()

arr\_date = two\_dates[0]

dept\_date = two\_dates[1]

print('Bedrooms free between ' + arr\_date + ' to ' + dept\_date + ':')

for r in reservation\_list:

if (date(dept\_date)<=date(r.arr\_date)) or (date(arr\_date)>=date(r.dept\_date)):

bedroom\_requests.append(str(r.room))

for b in bedroom\_list:

if str(b) not in reserved\_rooms(reservation\_list):

bedroom\_requests.append(str(b))

bedroom\_requests = list(set(bedroom\_requests))

for beds in bedroom\_requests:

print(beds)

#LO

def list\_occupied(line:str):

global bedroom\_list

bedroom\_requests = []

two\_dates = line[2:].split()

arr\_date = two\_dates[0]

dept\_date = two\_dates[1]

print('Bedrooms occupied between ' + arr\_date + ' to ' + dept\_date + ':')

for r in reservation\_list:

if not (date(dept\_date)<=date(r.arr\_date)) or (date(arr\_date)>=date(r.dept\_date)):

bedroom\_requests.append(str(r.room))

bedroom\_requests = list(set(bedroom\_requests))

for beds in bedroom\_requests:

print(beds)

CONCLUSION

1] ITS GIVES US AN IDEA ABOUT THE HOTEL MANAGEMENT SYSTEM

2] IT HELPS US TO EFFICENTLY MANAGE THE HOTEL WITHOUT ANY COMPLEXITY

3] IT GIVES US THE IDEA ABOUT USES OF PYTHON IN DIFFERENT FIELDS

REFRENCES

1] QUORA

2] WIKIPEDIA

3] GITHUB