|  |  |
| --- | --- |
| Code | Output |
| #passenger.py  no\_of\_passenger =int(input("Enter No of Passenger:"))  while no\_of\_passenger!=0 :      # print (no\_of\_passenger)      nb=input("Enter No of Baggage:")      no\_of\_baggage=int(nb)      while no\_of\_baggage!=0:          status=input("Enter Status of the baggage(Checked-C/Not Checked -N):")          print("Passenger",no\_of\_passenger,"--Baggage",no\_of\_baggage,"is",status)          no\_of\_baggage-=1      no\_of\_passenger-=1 | Enter No of Passenger:2  Enter No of Baggage:2  Enter Status of the baggage(Checked-C/Not Checked -N):C  Passenger 2 --Baggage 2 is C  Enter Status of the baggage(Checked-C/Not Checked -N):N  Passenger 2 --Baggage 1 is N  Enter No of Baggage:3  Enter Status of the baggage(Checked-C/Not Checked -N):C  Passenger 1 --Baggage 3 is C  Enter Status of the baggage(Checked-C/Not Checked -N):C  Passenger 1 --Baggage 2 is C  Enter Status of the baggage(Checked-C/Not Checked -N):C  Passenger 1 --Baggage 1 is C |
| def convert\_temp(temp, unit):      if unit=='C':          res\_temp=temp\*9/5+32      elif unit=='F':          res\_temp=(temp-32)\*5/9      else:          res\_temp="Incorrect Unit"      return res\_temp    print("Temp - ",convert\_temp(104,'A')) | Temp - 219.2 |
| #Break  for i in range(1,6):      if i==3:          break      print(i)  # Continue  for i in range(1,6):      if i==3:          continue      print(i) | 1  2  1  2  4  5 |
| def calculate\_total\_ticket\_cost(no\_of\_adults, no\_of\_children):      rate\_of\_adult=37550      rate\_of\_child = rate\_of\_adult/3      total\_ticket\_cost=no\_of\_adults\*rate\_of\_adult+no\_of\_children\*rate\_of\_child      total\_ticket\_cost+=total\_ticket\_cost\*0.07      total\_ticket\_cost-=total\_ticket\_cost\*0.10        return total\_ticket\_cost  #Provide different values for no\_of\_adults, no\_of\_children and test your program  total\_ticket\_cost=calculate\_total\_ticket\_cost(5,2)  print("Total Ticket Cost:",total\_ticket\_cost) | Total Ticket Cost: 204910.35 |
| # Program to find the ASCII value of the given character  c = 'a'  print("The ASCII value of '" + c + "' is", ord(c)) | The ASCII value of 'a' is 97 |
| # WAP to convert decimal to binary  for i in range(1,10001):      b\_no=s=""      temp=i      while i>0:          rem=str(i%2)          b\_no=b\_no+rem          i//=2      for j in b\_no:          s = j + s      print("Decimal value of ",temp,":",s) |  |
| # For Loop  # for i in 1,2,3,4,5:  #     print(i)  # sum=0  # for i in 1,2,3,4,5:  #     sum+=i  #     print(sum)  # Use of Range  # for i in range(1,6):  #     print(i)  # Another use of Range Function  # Three parameter 1= start 10 = end , 2 = increment value  for i in range(1,10,2):      # print(i)      print(i,end="") # prints in same line |  |
| # Python Program to find the L.C.M. of two input number  def compute\_lcm(x, y):     # choose the greater number     if x > y:         greater = x     else:         greater = y     while(True):         if((greater % x == 0) and (greater % y == 0)):          #    print(greater)             lcm = greater             break         greater += 1     return lcm  num1 = int(input("Enter first No : "))  num2 = int(input("Enter second No : "))  print("The L.C.M. is", compute\_lcm(num1, num2)) | Enter first No : 15  Enter second No : 17  The L.C.M. is 255 |
| import time  def tryit1():      for i in range(1,1000000):          x=time.strftime("%d/%m/%Y")    t = time.time()  tryit1()  print('duration:', time.time()-t) | duration: 4.709304332733154 |
| # Write a python program to convert one data capacity to another.  # Example:  # 4 terrabytes = ? GB  no=int(input("Enter no to convert : "))  print(no," terrabytes = ", (no\*1024) , " GB") | Enter no to convert : 71  71 terrabytes = 72704 GB |
| # list of list  my\_list =[["abc","def"],["ghi"],["alpha","beta"],2]  # print(my\_list[0])  # print(my\_list[2][0])  for sample\_list in my\_list:      for element in sample\_list:          print(element) | abc  def  ghi  alpha  beta  Traceback (most recent call last):  File "d:/STUDY/INFY/GENERIC/PROGRAM/demo.py", line 7, in <module>  for element in sample\_list:  TypeError: 'int' object is not iterable |
| my\_list =["alpha","beta","gama","delta"]  print(my\_list[2:]) #from 2 to last  print(my\_list[:]) # complete list  print(my\_list[:2]) # first and second element element of index 1 | ['gama', 'delta']  ['alpha', 'beta', 'gama', 'delta']  ['alpha', 'beta'] |
| my\_list=[0]\*5  for index in range(1,5):      my\_list[index]=(index-1)\*10  print(my\_list)  list\_of\_airline=["AI","SJ","JA","EM","BA"]  #slicing can be done using : 1:4  1= starting index, 4 = ending index  # it creates new list as per the given index  '''  [ 1  2  3  4  ]  [ 0  1  2  3  ] positive indexing  [ -4  -3  -2  -1  ] negative indexing  indexing always move from left to right  [1:4:2]  1 - starting index  4 - ending index  2 - increment  '''  new\_list=list\_of\_airline[1:2,3:4]  print(new\_list) | [0, 0, 10, 20, 30]  Traceback (most recent call last):  File "d:/STUDY/INFY/GENERIC/PROGRAM/demo.py", line 21, in <module>  new\_list=list\_of\_airline[1:2,3:4]  TypeError: list indices must be integers or slices, not tuple |
| counter=0  while counter<=9:      if counter%2==0:          pass      else:          print(counter,end=" ")      counter+=1 | 1 3 5 7 9 |
| n1=input()  n2=input()  num1=int(n1)  num2=int(n2)  while num1>=2:      if num1>num2:          num1=num1/2      else:          print(num1)          break | 5  6  5 |
| for number in 10,15:      for counter in range(1,3):          print(number\*counter, end=" ") | 10 20 15 30 |
| for num in 23,45,50,65,76,90:      if num%5!=0:          continue      if num%10==0:          print(num, end=" ")          continue      if num%3==0:          print(num,end=" ") | 45 50 90 |
| number=28  for num in range(25,30):      if number>num:          print(num)      else:          print(num)          break | 25  26  27  28 |
| def count\_names(name\_list):      count1=0      count2=0        #start writing your code here      for i in name\_list:          if(((i.find("at"))==1) and len(i)==3):              count1+=1          if(i.find("at"))>=0:              count2+=1      print("\_at -> ",count1)      print("%at% -> ",count2)  name\_list=["Hat","Cat","rabbit","matter"]  # name\_list=['Rat', 'saturday']  count\_names(name\_list) | \_at -> 2  %at% -> 3 |
| row1 = (101,"Dallas",3.5)  row2 = (102,"Atlanta",5.6)  row3 = (103,"Tokyo",9.8)  table = [row1,row2,row3]  print(table[0])  print(table[1])  print(table[2]) | (101, 'Dallas', 3.5)  (102, 'Atlanta', 5.6)  (103, 'Tokyo', 9.8) |
| # String Funstions  boarding\_call="Good Evening, this is the final call to AI passengers for the flight AI 466 which is planned to take off at 8.40A.M."  if(boarding\_call.startswith("Good Evening")):      print(boarding\_call.replace("Good Evening","Good Morning"))  if(boarding\_call.find("AI"))>=0:      print("Welcome to Air India.")  if(boarding\_call.endswith("A.M.")):      print("Passengers are requested to have their breakfast.")  a=boarding\_call.split(" ")  for i in a:      if(i.isdigit()):          print("Flight Number is specified to the passengers.")  print("Total number of times flight service name is specified in the boarding call:",boarding\_call.count("AI"))  message="Thank you all..Have a nice journey!"  print(message.upper())  print(message.lower()) | Good Morning, this is the final call to AI passengers for the flight AI 466 which is planned to take off at 8.40A.M.  Welcome to Air India.  Passengers are requested to have their breakfast.  Flight Number is specified to the passengers.  Total number of times flight service name is specified in the boarding call: 2  THANK YOU ALL..HAVE A NICE JOURNEY!  thank you all..have a nice journey! |
| def generate\_ticket(airline,source,destination,no\_of\_passengers):      ticket\_no=100      ticket\_number\_list=[]      while no\_of\_passengers>0:          temp\_ticket=airline+':'+source[:3]+':'+destination[:3]+':'+str((ticket\_no+1))          ticket\_no+=1          ticket\_number\_list.append(temp\_ticket)          no\_of\_passengers-=1      return ticket\_number\_list[-5:]  #Provide different values for airline,source,destination,no\_of\_passengers and test your program  print(generate\_ticket("AI","Bangalore","London",10)) | ['AI:Ban:Lon:106', 'AI:Ban:Lon:107', 'AI:Ban:Lon:108', 'AI:Ban:Lon:109', 'AI:Ban:Lon:110'] |
| #Creating a string  pancard\_number="AABGT6715H"  #Length of the string  print("Length of the PAN card number:", len(pancard\_number))  #Concatenating two strings  name1 ="PAN "  name2="card"  name=name1+name2  print(name)  print("Iterating the string using range()")  for index in range(0,len(pancard\_number)):      print(pancard\_number[index])    print("Iterating the string using keyword in")  for value in pancard\_number:      print(value)  print("Searching for a character in string")  if "Z" in pancard\_number:      print("Character present")  else:      print("Character is not present")  #Slicing a string  print("The numbers in the PAN card number:", pancard\_number[5:9])  print("Last but one 3 characters in the PAN card:",pancard\_number[-4:-1])  # pancard\_number[2]="A" #This line will result in an error, i.e., string is immutable  print(pancard\_number) | Length of the PAN card number: 10  PAN card  Iterating the string using range()  A  A  B  G  T  6  7  1  5  H  Iterating the string using keyword in  A  A  B  G  T  6  7  1  5  H  Searching for a character in string  Character is not present  The numbers in the PAN card number: 6715  Last but one 3 characters in the PAN card: 715  AABGT6715H |
| # String is another type of collection  # String is also immutable  # String Follow same concept of indexing  # String can be added  # String Declaration  # string\_one=""  # string\_two=''  # String Addition  string\_one="alpha"  string\_two='beta'  string\_one=string\_one+'delta'  print(string\_one)  #  when we add anything to a string new value is allocated to the string. | alphadelta  Enter a worda  Enter a worda  Enter a wordd  Enter a wordalpha |
| #WAP to take input untill alpha is given  a=""  while (a!="alpha"):      a=input("Enter a word") |  |
| gem\_list=["Emerald","Ivary","Jasper","Ruby","Garnet"]  price\_list=[1760,2119,1599,3920,3999]  required\_gem="Ruby"  required\_index=None  # for i in range(0,len(gem\_list)):  #     if gem\_list[i]==required\_gem:  #         required\_index=i  # print("Price of required Item is ",price\_list[required\_index])  reqd\_gems=["Ivary","Emerald","Garnet"]  reqd\_quantity=[3,10,12]  total\_price=0  for i in range(0,len(reqd\_gems)):      if(reqd\_gems[i] in gem\_list):          print("Found")      qty = reqd\_quantity[i]      for j in range(0,len(gem\_list)):          if reqd\_gems[i]==gem\_list[j]:              required\_index=j      total\_price += price\_list[required\_index]\*qty | Found  Found  Found |
| '''  b=(1,2,3)  tuple value cant be change  b=1, - tuple  b=(1) - int  '''  b=(1,2,3)  print(b[2]) |  |
| #set.py  # Problem Statement  # Try out the below code for usage of set and observe the results.  #list of passengers  passengers\_list=["George","Annie", "Jack","Annie","Henry", "Helen","Maria","George","Jack","Remo"]  #set function - removes the duplicates from the list and returns a set  unique\_passengers=set(passengers\_list)  print(unique\_passengers)  #creating a set  flight\_set={500,520,600,345,520,634,600,500,200,200}  print(flight\_set)  flights\_at\_src = ["AI230","BA944","EM395","AI704","BA944","AI704"]  flights\_at\_dest = ["SI107","AI034","EM395","AI704","BA802","SI236"]  print(flights\_at\_src)  print(flights\_at\_dest)  #Creating list of unique flights at source and destination  uniq\_src\_flights = set(flights\_at\_src)  uniq\_dest\_flights = set(flights\_at\_dest)  print(uniq\_src\_flights)  print(uniq\_dest\_flights)  #setA-setB -> Gives the elements that are only in setA  #List of flights only at source airport  flights\_only\_at\_src = uniq\_src\_flights-uniq\_dest\_flights  print(flights\_only\_at\_src)  #setA&setB -> Gives the common elements between setA and setB  #List of flights common to source and destination airports  common\_flights=uniq\_src\_flights&uniq\_dest\_flights  print(common\_flights)  #setA|setB -> merges setA and setB after removing duplicates  #List of all flights at source and destination airports  all\_flights=uniq\_src\_flights|uniq\_dest\_flights  print(all\_flights) | {'Annie', 'Helen', 'Remo', 'George', 'Jack', 'Henry', 'Maria'}  {520, 200, 500, 600, 345, 634}  ['AI230', 'BA944', 'EM395', 'AI704', 'BA944', 'AI704']  ['SI107', 'AI034', 'EM395', 'AI704', 'BA802', 'SI236']  {'AI230', 'AI704', 'BA944', 'EM395'}  {'EM395', 'AI704', 'SI107', 'BA802', 'AI034', 'SI236'}  {'AI230', 'BA944'}  {'AI704', 'EM395'}  {'EM395', 'AI230', 'AI704', 'BA944', 'SI107', 'BA802', 'AI034', 'SI236'} |
| #Creating a dictionary  crew\_details={              "Pilot":"Kumar",              "Co-Pilot":"Raghav",              "Head-Strewardess":"Malini",              "Stewardess":"Mala"  }  print(crew\_details["Pilot"])  print("\nIterating the dictionary using items function")  for key,value in crew\_details.items():      print(key,":",value)  #Usually while working with dictionary, you will be interested in specific values.  #Let’s find the value of all pilots from crew\_details.  print("\nIterating the dictionary using keyword 'in'")  for key in crew\_details:      if(key=="Pilot" or key=="Co-Pilot"):          print(crew\_details[key])  #Note: Dictionary being unordered, the order of the values being displayed may vary during each execution of the above for loop.  #Dictionaries are mutable  crew\_details["Pilot"]="James" # Here the value for key "Pilot" is being updated to "James"  print("\nAfter modifying the value of Pilot:", crew\_details["Pilot"])  print("------------------------------------------------------------------")  print("Before update:")  # Usage of get method()  print("Co-pilot:",crew\_details.get("Co-Pilot"))  #Usage of update method()  crew\_details.update({"Flight Attendant":"Jane", "Co-pilot":"Henry"})  print("\nAfter update:")  print("Co-pilot:",crew\_details.get("Co-pilot"))  print("Flight Attendant:",crew\_details["Flight Attendant"]) | Iterating the dictionary using items function  Pilot : Kumar  Co-Pilot : Raghav  Head-Strewardess : Malini  Stewardess : Mala  Iterating the dictionary using keyword 'in'  Kumar  Raghav  After modifying the value of Pilot: James  ------------------------------------------------------------------  Before update:  Co-pilot: Raghav  After update:  Co-pilot: Henry  Flight Attendant: Jane |
| '''  Given a string containing uppercase characters (A-Z), compress the string using Run Length encoding. Repetition of character has to be replaced by storing the length of that run.  Write a python function which performs the run length encoding for a given String and returns the run length encoded String.  '''  def encode(message):      counter = 1      result = ""      previousLetter = message[0]      if len(message)==1:        return str(1) + message[0]      for i in range(1,len(message),1):          if not message[i] == previousLetter:              result += str(counter) + message[i-1]              previousLetter = message[i]              counter = 1          else:              counter += 1          if i == len(message)-1:                  result += str(counter) + message[i]      return result  #Provide different values for message and test your program  encoded\_message=encode("ABBBBCCCCCCCCAB")  print(encoded\_message) | 1A4B8C1A1B |

Program – Set of instructions to perform a specific task by computer.

Level of program

1. High Level
2. Low Level – directly interact with hardware

Searching

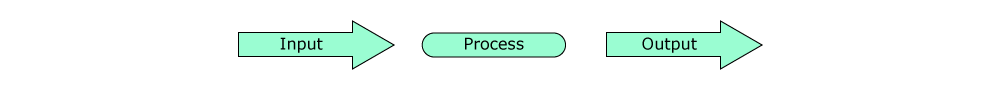
Counting

Optimization – limited resources due to which optimization needed

Sorting

Decision

How to solve problem

The solution to any problem involves the below three aspects:

Algorithm & its Representation

In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.

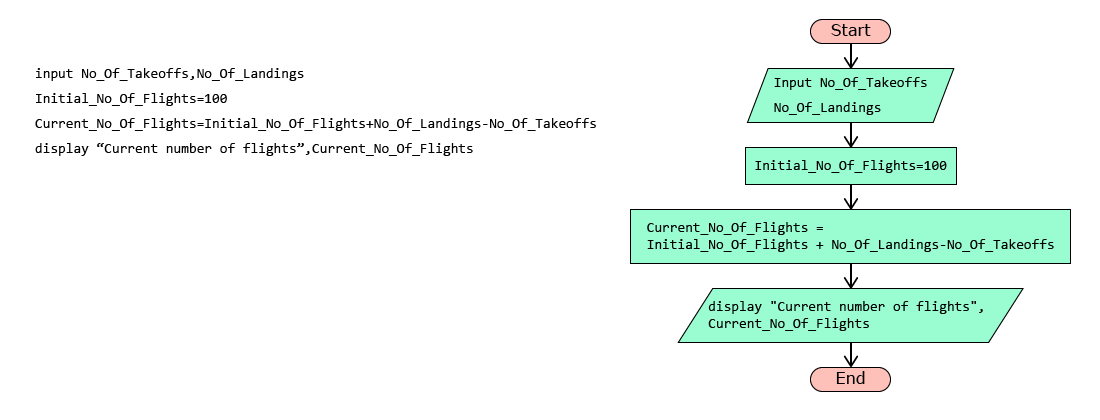
Representation of an algorithm

Algorithm should be represented from our mental thoughts into a form which others can understand. There are primarily two ways of representing an algorithm:

* Flow chart: Diagrammatic way of representing the algorithm.
* Pseudo-code: Representation of the algorithm in a way that is in between a program and normal English.

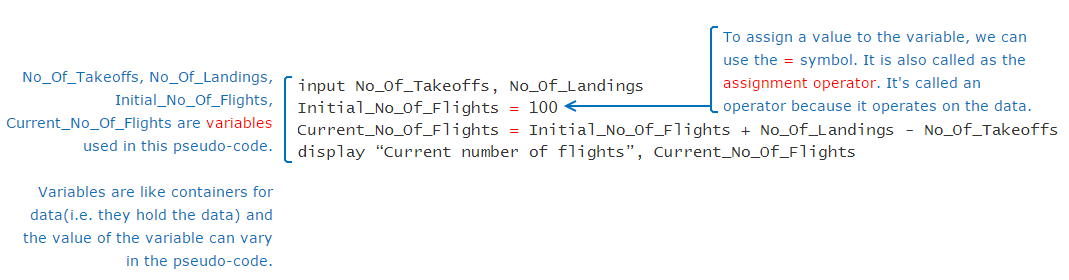
For example, here is a pseudo-code and a flowchart.

**Pseudo-code:**                                                                                                                                             **FlowChart:**



Pseudo-code cannot be executed by a computer. It is just a representation of an algorithm for us to understand.

Variables & Operators



Variable - Container for any value

Like assignment operator, there are other operators also which can be used to perform various operations.  
Arithmetic operators: Used for performing arithmetic operations

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **+** | Addition |
| **-** | Subtraction |
| **\*** | Multiplication |
| **/** | Division |
| **%** | Modulus |

Relational operators: Also known as comparison operators, are used to compare values. Result of a relational expression is always either true or false.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **==** | Equal to |
| **<** | Less than |
| **>** | Greater than |
| **<=** | Less than or equal to |
| **>=** | Greater than or equal to |
| **!=** | Not equal to |

Logical operators are used to combine one or more relational expressions.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| AND | Result will be true, if both the expressions are true. If any one or both the expressions are false, the result will be false |
| OR | Result will be true, even if one of the expression is true. If both the expressions are false, the result will be false |
| NOT | If the expression is true, result will be false and vice versa |

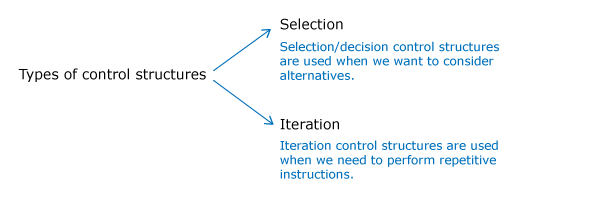
If A and B are two relational expressions, say A = (Num1>2000), B= (Num2>100), the result of combining A and B using logical operator is based on the result of A and B as shown below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A AND B** |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

Decision Constructs

Flow of control

In a pseudo-code, typically the instructions are performed one by one or line by line. But there may be situations when all the statements in a pseudo-code are not performed. Parts of the pseudo-code which change the flow of instructions or in other word, change the flow of control are called as control structures.

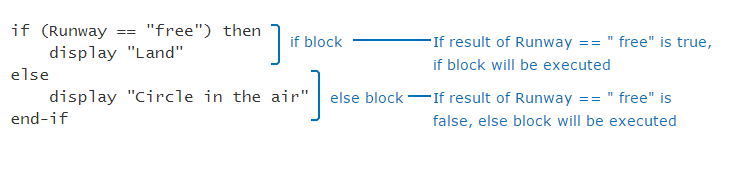


Selection using if statement

ATC takes lot of decisions as part of its air traffic control operations.

For example, if a flight is approaching the runway, ATC has to check if the runway is free. If the runway is not free, then the flight should not land immediately. It should circle in the air and wait for further instructions from the ATC.

Such decision making process can be conveniently represented in a pseudo-code using an if statement.



Sometimes, ATC may have more than one alternatives for a given situation. For example, if the runway is free, the flight can land. But if the flight has less fuel, then it should be allowed an emergency landing. Otherwise, it should circle in the air.

if (Runway=="free") then

   display "Land"

else if (Fuel\_Status=="low") then

   display "Emergency landing"

else

   display "Circle in the air"

end-if

Iteration Constructs

Immigration check needs to be done for all the passengers in the flight. Suppose the flight had only 5 passengers, the pseudo-code can be written as follows:

display "Flight has landed"

display "Proceed for Immigration Check"

Passenger\_Count=1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Passenger\_Count=Passenger\_Count+1

display "Immigration check done for passenger,", Passenger\_Count

Sometimes, we may want to execute some statements specific (known) number of times as in the case of immigration check. In such cases we can use a **for loop**as shown below.

No\_Of\_Passengers=5

for(Passenger\_Count=1,Passenger\_Count<=No\_Of\_Passengers,Passenger\_Count=Passenger\_Count+1)

     display "Immigration check done for passenger,", Passenger\_Count

end-for

When we want to repeatedly execute a statement as long as a condition is met, we can use the iteration statement called as **while loop**.

display "The flight has landed"

display "Immigration check done"

display " Collect the baggage from the conveyor belt"

Baggage\_Count=150

while(Baggage\_Count>0) do

    input No\_Of\_Baggage\_Picked

    Baggage\_Count=Baggage\_Count-No\_Of\_Baggage\_Picked

end-while

Go through the below two pseudo-codes and guess the output.

|  |  |
| --- | --- |
| **Pseudo-code 1 :** | **Pseudo-code 2 :** |
| 1. Counter=5 2. while(Counter>=5) do 3. display Counter 4. Counter = Counter + 1 5. end-while | 1. input Limit 2. for(Counter=5, Counter<=Limit, Counter=Counter-1) 3. display Counter 4. end-for   Assume that the input value provided to variable Limit is |

Beware of infinite loops. The logic that we are writing in loops should ensure that the loop will terminate in finite number of iterations.

What is the outcome of the following pseudo-code?

input Counter

while(Counter<5) do

  Counter=Counter+1

  display Counter

end-while

Assume that the input value provided to variable, Counter is 1.

2,3,4,5

2,3,4

1,2,3,4

1,2,3,4,5

Programming languages

Pseudo-code helped us to represent the algorithms and learn few basics of programming. But to instruct the computer we need to write a program in a programming language. There are many languages available in which we can write our programs.

Different languages are created for different purposes. This involves trade-offs. For example a large flight can carry many passengers, but also consumes lot of fuel. Similarly different languages have different advantages.

Datatypes

Data

We know that a program works on data or values. Some examples of values are 1, 1.0 and "Hello".For a program to use a data, it must be stored in a memory location. The values are stored in binary form in computer memory.

How do we know how many bits are required to represent a value or how much space it needs in memory?

This is determined by the data type of the value and the programming language. e.g. number, string etc. Thus every value will have a type.

Datatypes

Data Type determines the operations that can be performed on a value. For e.g. we can perform operations like addition, multiplication, division, subtraction etc. on numerical data types. We can concatenate, convert case, extract substring etc. on string data types.

1. print(1 + 2)
2. print("Hello " + "World")
3. print(True and False)
4. print(4.0 / 2.0)

The above program uses 8 values of different data types.

A program may have data belonging to different types. Common data types used in programming are:

|  |  |  |
| --- | --- | --- |
| **Category** | **Data Type** | **Example** |
| Numeric | int | 123 |
| long | 1237126381763817 |
| Numeric with decimal point | float | 123.45 |
| double | 123123.32345324 |
| Alphanumeric | char | A |
| String | hello |
| Boolean | boolean | True, False |

Python programming language supports the following datatypes:

|  |  |
| --- | --- |
| **Category** | **Python** |
| Numeric | int |
| long |
| complex |
| Numeric with decimal point | float |
| Alphanumeric | String |
| Boolean | boolean |

### **Problem Statement**

In Python, the data type of a value can be identified by using **type**(value).

Try out the below program and observe the results.

print(type(3))

print(type("Hello World"))

print(type(False))

print(type(2.0))

Output:

<class 'int'>  
<class 'str'>  
<class 'bool'>  
<class 'float'>

Variables



If everything is a value, then what is a variable?

A variable is a name that is assigned to a value. It is done so that we can refer to that value at some later point in the program.

Let us see the usage of values and variables in a program.

* During input, we receive values and assign them to variables.
* During processing, we perform operations on values and variables to generate more values. These computed values are also assigned to variables.
* During output, we display the computed values in variables to the end users.

Let’s have a look at the python program to display the number of landings and number of takeoffs in an airport:

Note: In Python, print() can be used to display output in the console

### **Problem Statement**

Try out the below program. This is the program for a pseudo-code which we had discussed earlier.  
Change the value of no\_of\_landings, no\_of\_takeoffs and initial\_no\_of\_flights, execute and observe the output.

no\_of\_landings=356

no\_of\_takeoffs=245

initial\_no\_of\_flights=100

current\_no\_of\_flights=initial\_no\_of\_flights+no\_of\_landings-no\_of\_takeoffs

print("Current number of flights:",current\_no\_of\_flights)

Output:

Current number of flights: 211

Static typed language fast – java

Dynamic typed language slow - python

Variables & Datatypes

Let’s see how can we associate a data type with a variable.

|  |
| --- |
| **Python** |
| num=100  msg="Hello" |

Why do you think the data type is not mentioned in Python?

Static and dynamic typing

Languages like Python are dynamically typed whereas C,Go etc are statically typed.

**Dynamic Typing**is a technique in some languages where depending on how a value is used, the data type of the variable is dynamically and automatically assigned. Consider the below code in Python,

num=65 #Line 1

num="A" #Line 2

In Line 1, variable num is considered to be of type int and in Line 2, its type is reassigned to String.

**Static Typing** is used in some languages where the data type has to be declared before a variable is used. Consider the below code in Go,

var num int=65; //Line 1

num="A"; //Line 2

var name string = "A"; //Line3

Here, Line 1 is a valid statement which declares a variable num of type int. But Line 2 is invalid as we cannot assign a string value to variable num which is already declared to be of type int. Line 3 is a valid statement where name is declared and used as a string.

Reserved words

Any name can be given to a variable however, we cannot use some of the built-in keywords of the language. These keywords are known as **reserved words**. Some of the reserved words in Python are:

|  |  |
| --- | --- |
| **Python** | if, else, for, while, def, print, raise, try, except |

Storing data

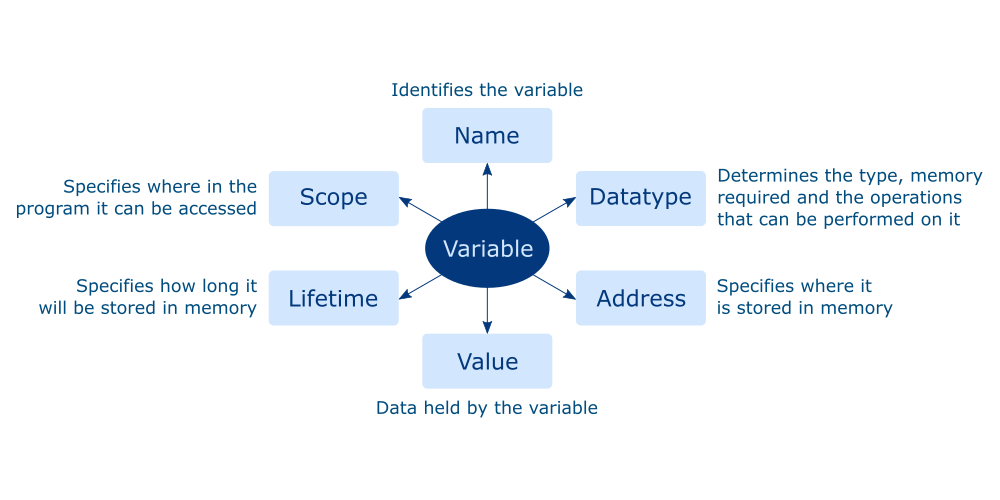
If you put a small product in a big box, you may end up wasting space. Similarly if you put a big product in a small box you may end up damaging the product.

Just like product is placed in a box, data occupies memory. Some data need more memory whereas some other data require less memory based on the data type.

In Python language, automatically creates just the right memory needed – neither less nor more.

Variable and its dimensions

We have seen that a variable will have a name, value, type and it will occupy memory. Apart from these, it has two more dimensions – scope and lifetime. Thus we can say that any variable will have the following six dimensions.



Operators

As we discussed in pseudo-code, operators help to perform an operation.

Some of the most common operators used in Python are listed below.

|  |  |
| --- | --- |
| **Common Operators** | **Python** |
| Arithmetic Operators | +,-,\*,/, %,// |
| Relational Operators | ==,!=,>,<,>=,<= |
| Assignment Operators | =,+=,-=,\*=,/=,%= |
| Logical Operators | and,or,not |

Note: In Python, // indicates integer division.

         Example: 11//2=5

True or False

The result of a relational or logical expression is always a boolean (true or false). Apart from the boolean values, other values can also be used to represent a true or false value.

For example, in python, value zero is considered to be equivalent to false.

The common false values in Python are given below. Any other value is considered to be true.

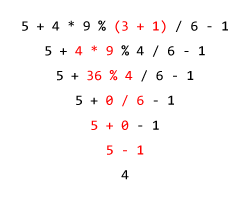
|  |
| --- |
| **Python** |
| None |
| False |
| 0 |
| ''/"" (Empty string - two single quotes/double quotes) |

Precedence of operators

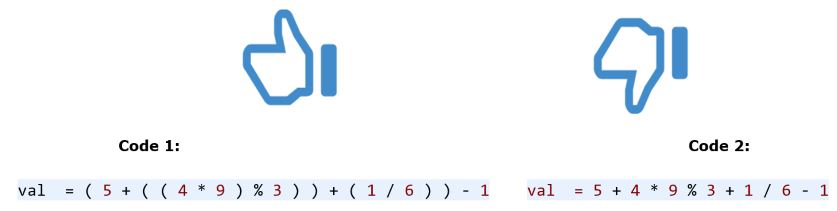
What do you think is the output of **5+4\*9%(3+1)/6-1**?  
How do you think the result of this expression is computed?

It is done based on the precedence of the operators. Precedence of an operator can be identified based on the rule - BODMAS. Brackets followed by Orders (Powers, Roots), followed by modulo, Division and Multiplication, followed by Addition and Subtraction.

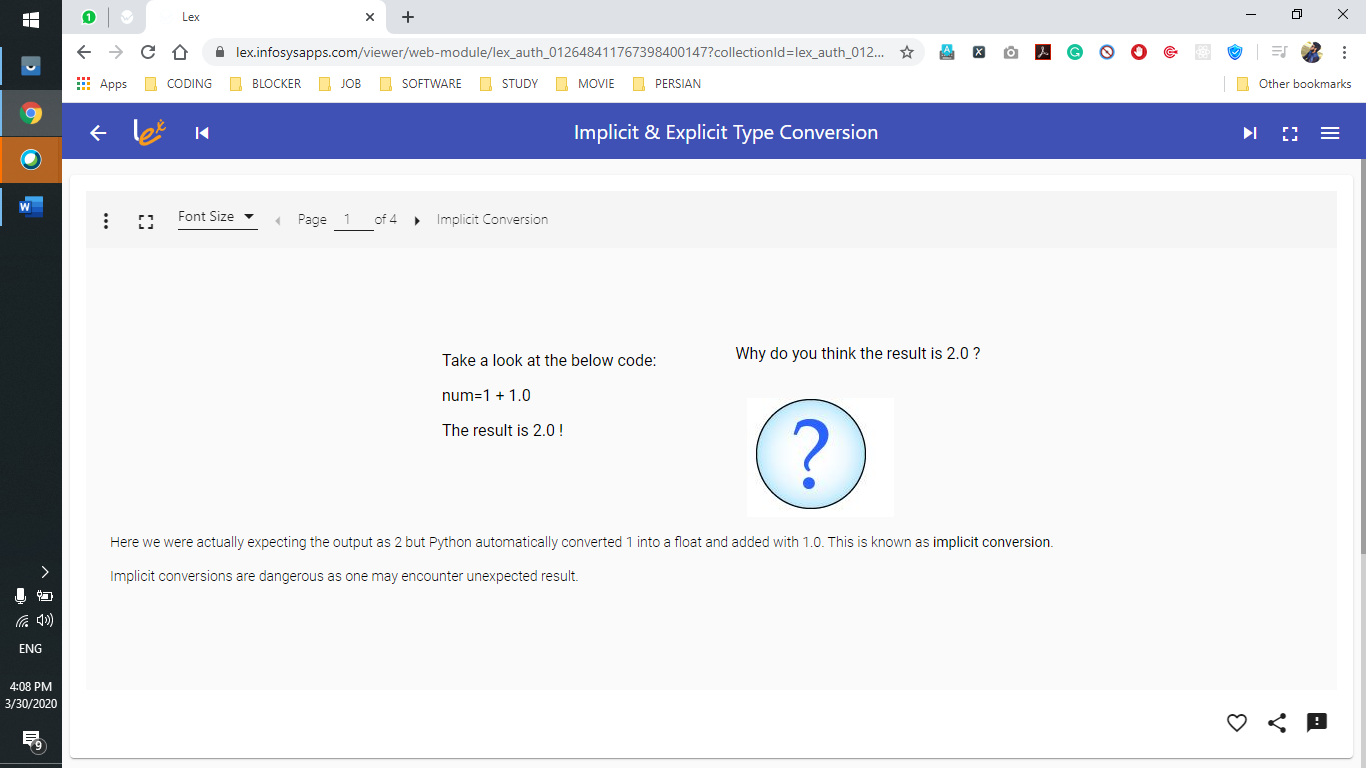
1. Brackets have the highest precedence followed by orders.
2. Modulo, Division and Multiplication have the same precedence. Hence if all appear in an expression, they are evaluated from Left to Right.
3. Addition and Subtraction have the same precedence. Hence if both appear in an expression, they are evaluated from Left to Right.

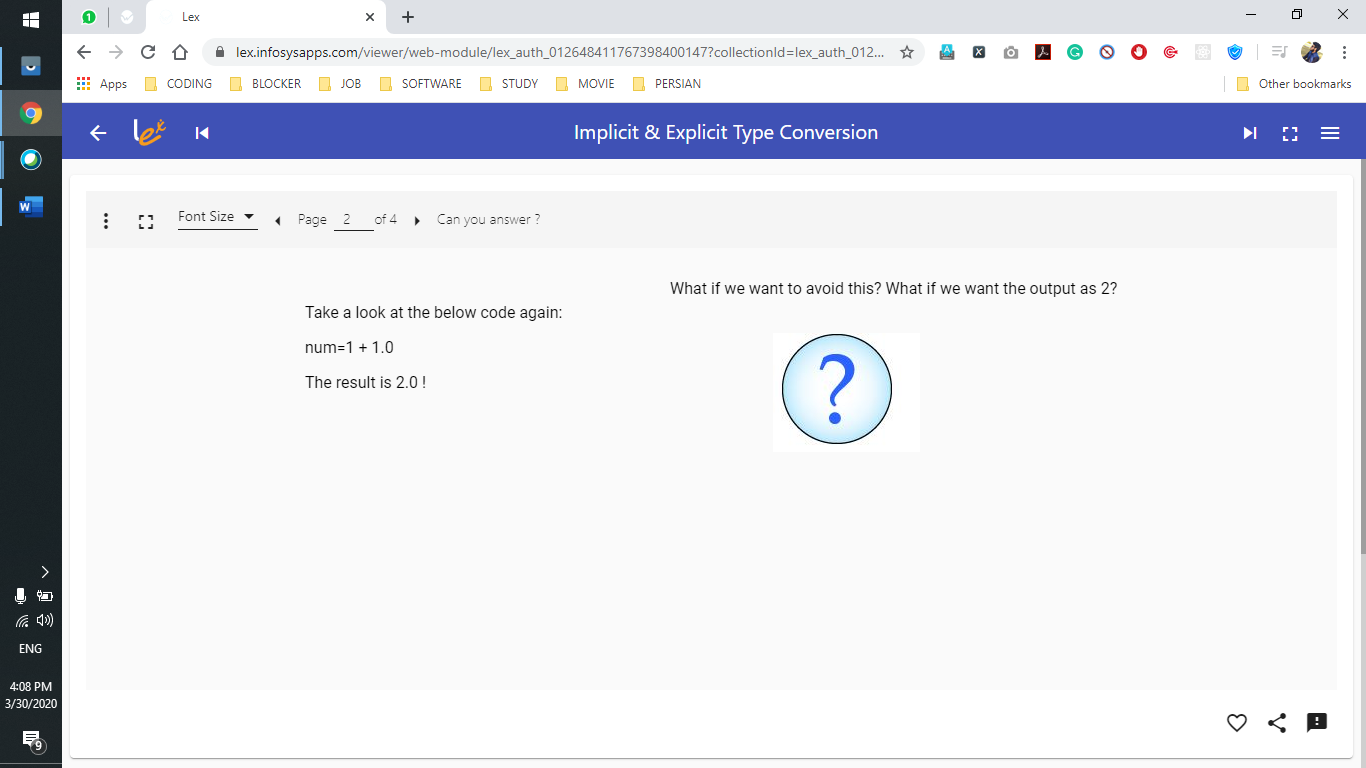


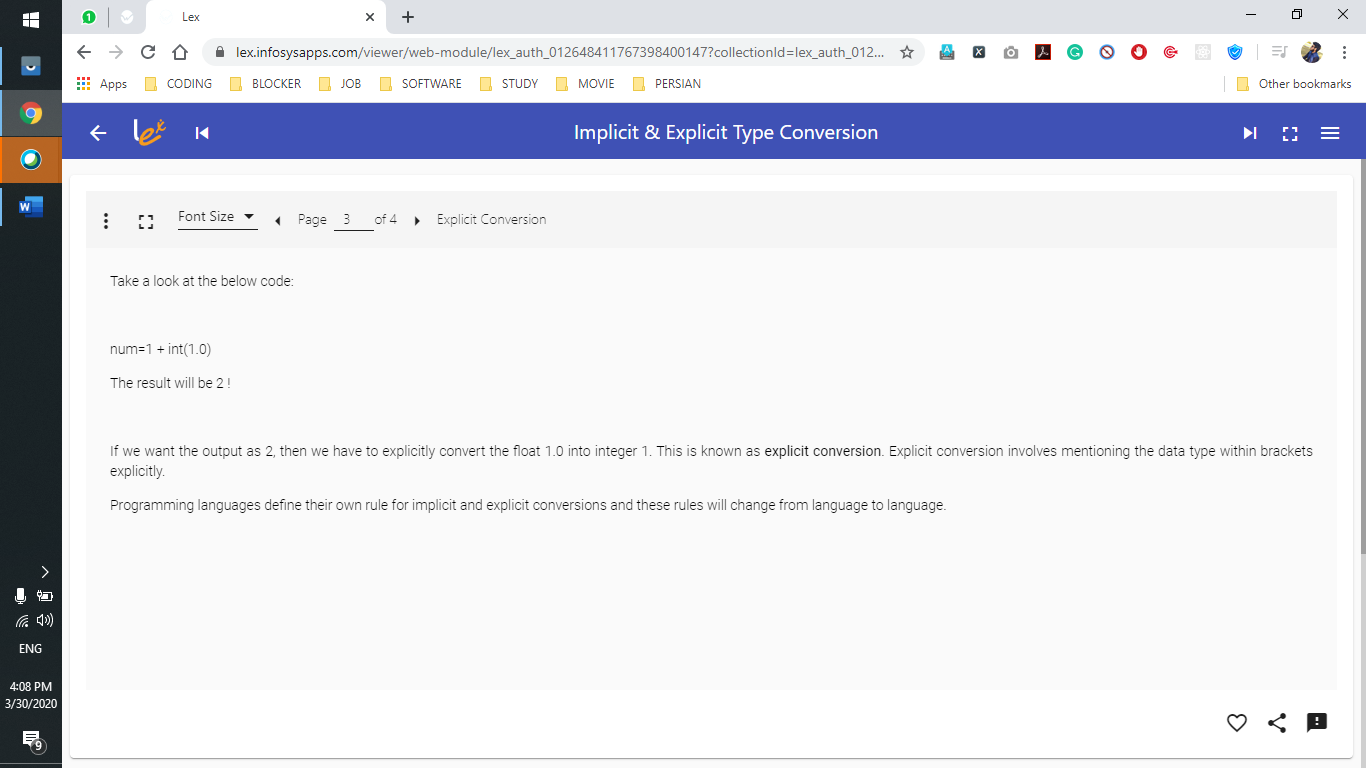
Though its not compulsory to have brackets to denote the precedence of operators in an expression, it is always preferred to have brackets as it makes the expression readable and brings in a lot more clarity.

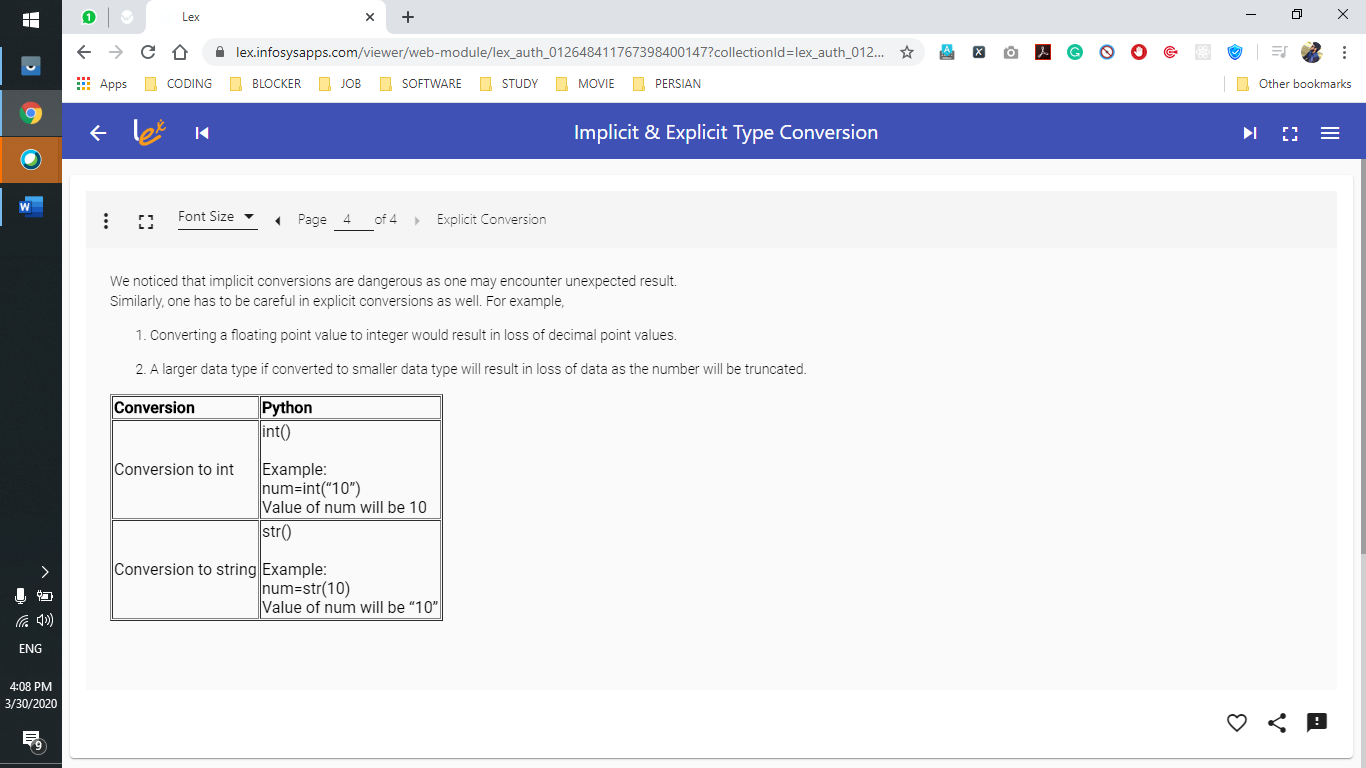


Implicit & Explicit Type Conversion









DAY 2 March 31st

String

In a program, not all values will be numerical. We will also have alphabetical or alpha numerical values. Such values are called strings.

**Example:** “Hello World”, pancard number - “AABGT6715H” , name – “Arjun”, ticket number – “BA1045”   
Each value in a string is called a **character**. Just like list elements, we can access the characters in a string based on its index position.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| String | "AABGT6715H" | | | | | | | | | |
| Character | A | A | B | G | T | 6 | 7 | 1 | 5 | H |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

In Python, string is a data type and anything enclosed in a single quote or double quote is considered to be a string. All the remaining operations are similar to lists. But like tuple, strings are also **IMMUTABLE.**

# String is another type of collection

# String is also immutable

# String Follow same concept of indexing

# String can be added

# String Declaration

# string\_one=""

# string\_two=''

# String Addition

string\_one="alpha"

string\_two='beta'

string\_one=string\_one+'delta'

print(string\_one)

#  when we add anything to a string new value is allocated to the string.

Try out - String  
#Creating a string

pancard\_number="AABGT6715H"

#Length of the string

print("Length of the PAN card number:", len(pancard\_number))

#Concatenating two strings

name1 ="PAN "

name2="card"

name=name1+name2

print(name)

print("Iterating the string using range()")

for index in range(0,len(pancard\_number)):

    print(pancard\_number[index])

print("Iterating the string using keyword in")

for value in pancard\_number:

    print(value)

print("Searching for a character in string")

if "Z" in pancard\_number:

    print("Character present")

else:

    print("Character is not present")

#Slicing a string

print("The numbers in the PAN card number:", pancard\_number[5:9])

print("Last but one 3 characters in the PAN card:",pancard\_number[-4:-1])

# pancard\_number[2]="A" #This line will result in an error, i.e., string is immutable

print(pancard\_number)

Exercise on String - Level 2

### **Problem Statement**

Write a python program to generate the ticket numbers for specified number of passengers traveling in a flight as per the details mentioned below:  
The ticket number should be generated as **airline:src:dest:number**  
where

1. Consider AI as the value for airline
2. src and dest should be the first three characters of the source and destination cities.
3. number should be auto-generated starting from 101

The program should return the list of ticket numbers of last five passengers.  
**Note:** If passenger count is less than 5, return the list of all generated ticket numbers.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| airline = AI source = Bangalore destination = London no\_of\_passengers = 10 | ['AI:Ban:Lon:106', 'AI:Ban:Lon:107', 'AI:Ban:Lon:108', 'AI:Ban:Lon:109', 'AI:Ban:Lon:110'] |
| airline = BA source = Australia destination = France no\_of\_passengers = 2 | ['BA:Aus:Fra:101', 'BA:Aus:Fra:102'] |

def generate\_ticket(airline,source,destination,no\_of\_passengers):

    ticket\_no=100

    ticket\_number\_list=[]

    while no\_of\_passengers>0:

        temp\_ticket=airline+':'+source[:3] +':'+destination[:3] +':'+str(ticket\_no+1)

        ticket\_no+=1

        ticket\_number\_list.append(temp\_ticket)

        no\_of\_passengers-=1

    return ticket\_number\_list[-5:]

#Provide different values for airline,source,destination,no\_of\_passengers and test your program

print(generate\_ticket("AI","Bangalore","London",10))

Try out - String Functions

String data type in Python has many inbuilt functions which make it easier to work with strings.

Consider the string, name="Raghav".

|  |  |  |
| --- | --- | --- |
| **Function** | **Output** | **Explanation** |
| name.count("a") | 2 | Returns the count of a given set of characters. Returns 0 if not found |
| name.replace("a","A") | RAghAv | Returns a new string by replacing a set of characters with another set of characters. It does not modify the original string |
| name.find("a") | 1 | Returns the first index position of a given set of characters |
| name.startswith("Ra") | True | Checks if a string starts with a specific set of characters, returns true or false accordingly. |
| name.endswith("ha") | False | Checks if a string ends with a specific set of characters, returns true or false accordingly. |
| name.isdigit() | False | Checks if all the characters in the string are numbers, returns true or false accordingly. |
| name.upper() | RAGHAV | Converts the lowercase letters in string to uppercase |
| name.lower() | raghav | Converts the uppercase letters in string to lowercase |
| name.split("a") | ['R', 'gh', 'v'] | Splits string according to delimiter and returns the list of substring. Space is considered as the default delimiter. |

Try out the code which uses the String built-in functions and observe the results.

# String Funstions

boarding\_call="Good Evening, this is the final call to AI passengers for the flight AI 466 which is planned to take off at 8.40A.M."

if(boarding\_call.startswith("Good Evening")):

    print(boarding\_call.replace("Good Evening","Good Morning"))

if(boarding\_call.find("AI"))>=0:

    print("Welcome to Air India.")

if(boarding\_call.endswith("A.M.")):

    print("Passengers are requested to have their breakfast.")

a=boarding\_call.split(" ")

for i in a:

    if(i.isdigit()):

        print("Flight Number is specified to the passengers.")

print("Total number of times flight service name is specified in the boarding call:",boarding\_call.count("AI"))

message="Thank you all..Have a nice journey!"

print(message.upper())

print(message.lower())

OUTPUT

Good Morning, this is the final call to AI passengers for the flight AI 466 which is planned to take off at 8.40A.M.

Welcome to Air India.

Passengers are requested to have their breakfast.

Flight Number is specified to the passengers.

Total number of times flight service name is specified in the boarding call: 2

THANK YOU ALL..HAVE A NICE JOURNEY!

thank you all..have a nice journey!

Choosing Between List, Tuple and String

### **Problem Statement**

* Represent the data in the table below in Python such that the data in a row cannot be changed but rows can be shifted up or down in the table
* Strings are used when you want a sequence of characters to act as a unit of information.
* Lists and tuples when you want to have a sequence of individual elements.

#string4.py

row1 = (101,"Dallas",3.5)

row2 = (102,"Atlanta",5.6)

row3 = (103,"Tokyo",9.8)

table = [row1,row2,row3]

print(table[0])

print(table[1])

print(table[2])

Quiz – String

### **Q1 of 3**

### Consider the following list of pan card numbers: pancard\_list=["AABGT6715H", "UFFAC4352T", "IFSBD9163K", "JOOEC1225H","RWXAFE187B"]

What is the output of the below two print statements?

print(pancard\_list[3][6], end=" ")

print(pancard\_list[4][3:])

2 AFE187B

### **Q2 of 3**

What is the output of the code given below?

message="welcome to Mysore"

word=message[-7:]

if(word=="mysore"):

    print("got it")

else:

    message=message[3:14]

    print(message)

come to Mys

### **Q3 of 3**

What is the output of the below code?

song="JINGLE Bells jingle Bells Jingle All The Way"

song.upper()

song\_words=song.split()

count=0

for word in song\_words:

if(word.startswith("jingle")):

count=count+1

print(count)

1

Exercise on String Functions - Level 1

### **Problem Statement**

Write a python program which displays the count of the names that matches a given pattern from a list of names provided.

Consider the pattern characters to be:

1. "\_ at" where "\_" can be one occurrence of any character

2. "%at%" where "%" can have zero or any number of occurrences of a character

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| [Hat, Cat, Rabbit, Matter] | \_at -> 2 %at% -> 3 |

def count\_names(name\_list):

    count1=0

    count2=0

    #start writing your code here

    for i in name\_list:

        if(((i.find("at"))==1) and len(i)==3):

            count1+=1

        if(i.find("at"))>=0:

            count2+=1

    print("\_at -> ",count1)

    print("%at% -> ",count2)

name\_list=["Hat","Cat","rabbit","matter"]

# name\_list=['Rat', 'saturday']

count\_names(name\_list)

Set

A set is an unordered group of values with no duplicate entries. Set can be created by using the keyword set or by using curly braces {}. set function is used to eliminate duplicate values in a list.

|  |  |  |
| --- | --- | --- |
| Creating a set | flight\_set={500,520,600,345,520,634,600,500,200,200} | Removes the duplicates from the given group of values to create the set |
| Eliminating duplicates from a list | passengers\_list=["George", "Annie", "Jack", "Annie", "Henry", "Helen", "Maria", "George", "Jack", "Remo"] unique\_passengers=set(passengers\_list) | set function - removes the duplicates from the list and returns a set |
| Common elements between setA and setB | setA & setB | Creates a new set which has common elements from setA and setB |
| Elements that are only in setA | setA - setB | Creates a new set which has only unique elements of setA |
| Merges elements of setA and setB | setA | setB | Creates a new set which has all the elements of setA and setB |

### **Problem Statement**

Try out the below code for usage of set and observe the results.

#set.py

# Problem Statement

# Try out the below code for usage of set and observe the results.

#list of passengers

passengers\_list=["George","Annie", "Jack","Annie","Henry", "Helen","Maria","George","Jack","Remo"]

#set function - removes the duplicates from the list and returns a set

unique\_passengers=set(passengers\_list)

print(unique\_passengers)

#creating a set

flight\_set={500,520,600,345,520,634,600,500,200,200}

print(flight\_set)

flights\_at\_src = ["AI230","BA944","EM395","AI704","BA944","AI704"]

flights\_at\_dest = ["SI107","AI034","EM395","AI704","BA802","SI236"]

print(flights\_at\_src)

print(flights\_at\_dest)

#Creating list of unique flights at source and destination

uniq\_src\_flights = set(flights\_at\_src)

uniq\_dest\_flights = set(flights\_at\_dest)

print(uniq\_src\_flights)

print(uniq\_dest\_flights)

#setA-setB -> Gives the elements that are only in setA

#List of flights only at source airport

flights\_only\_at\_src = uniq\_src\_flights-uniq\_dest\_flights

print(flights\_only\_at\_src)

#setA&setB -> Gives the common elements between setA and setB

#List of flights common to source and destination airports

common\_flights=uniq\_src\_flights&uniq\_dest\_flights

print(common\_flights)

#setA|setB -> merges setA and setB after removing duplicates

#List of all flights at source and destination airports

all\_flights=uniq\_src\_flights|uniq\_dest\_flights

print(all\_flights)

{'Annie', 'Helen', 'Remo', 'George', 'Jack', 'Henry', 'Maria'}

{520, 200, 500, 600, 345, 634}

['AI230', 'BA944', 'EM395', 'AI704', 'BA944', 'AI704']

['SI107', 'AI034', 'EM395', 'AI704', 'BA802', 'SI236']

{'AI230', 'AI704', 'BA944', 'EM395'}

{'EM395', 'AI704', 'SI107', 'BA802', 'AI034', 'SI236'}

{'AI230', 'BA944'}

{'AI704', 'EM395'}

{'EM395', 'AI230', 'AI704', 'BA944', 'SI107', 'BA802', 'AI034', 'SI236'}

Quiz - Set

### **Q1 of 1**

Given listA = [1,2,3,4,5,5,6,6,7,7,7,8,8,8,8]

What will be the output of set(listA)?

1,2,3,4,5,6,7,8

Dictionary

Flight count

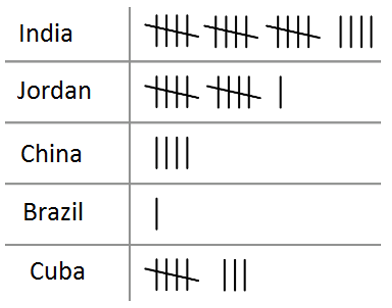
Suppose these are the 5 countries to which flights operate from an airport.



We want to know the country to which maximum flights have departed on a day.

How can we do this?

We can maintain a count for each country. As and when a flight departs from the airport, count for the country to which it has departed can be increased by one. At the end of the day, we just need to pick the country with the maximum count from this.

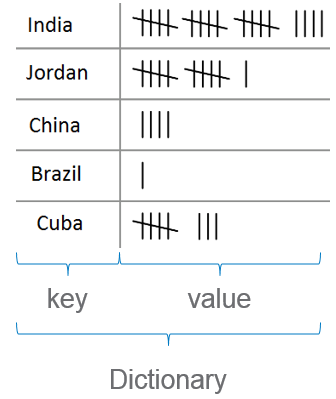


Here we needed two sets of data to answer our question – countries and their count.

Independently they do not make sense. We need to store them as key-value pairs.

In programming, this can be done using a collection known as **dictionary** which allows to store key-value pairs where each key is unique.

One of the advantage of using dictionary is that it allows very fast search for value based on key.



A dictionaries can be used to store an unordered collection of key-value pairs. The key should be unique and can be of any immutable data type. Like lists, dictionaries are mutable. Let’s now understand how a dictionary is implemented in Python.

|  |  |  |
| --- | --- | --- |
| Creating a dictionary | crew\_details= { "Pilot":"Kumar", "Co-pilot":"Raghav", "Head-Strewardess":"Malini", "Stewardess":"Mala" } | First element in every pair is the key and the second element is the value. |
| Accessing the value using key | crew\_details["Pilot"] | This will return the corresponding value for the specified key |
| Iterating through the dictionary | for key,value in crew\_details.items():      print(key,":",value) | items function gives both key and value, which can be used in a for loop. |

Dictionary in Python also have many inbuilt functions.

| **Function** | **Output** | **Explanation** |
| --- | --- | --- |
| crew\_details.get("Pilot") | Kumar | Returns the value for given key. If the given key is not found, returns None |
| crew\_details.update({"Flight Attendant":"Jane", "Co-pilot":"Henry"}) | No output, dictionary will be updated | Updates the dictionary with the given key-value pairs. If a key-value pair is already existing, it will be overwritten, otherwise it will be added to the dictionary |

Try out – Dictionary

### **Problem Statement**

Try out the below code and observe the results.

#Creating a dictionary

crew\_details={

            "Pilot":"Kumar",

            "Co-Pilot":"Raghav",

            "Head-Strewardess":"Malini",

            "Stewardess":"Mala"

}

print(crew\_details["Pilot"])

print("\nIterating the dictionary using items function")

for key,value in crew\_details.items():

    print(key,":",value)

#Usually while working with dictionary, you will be interested in specific values.

#Let’s find the value of all pilots from crew\_details.

print("\nIterating the dictionary using keyword 'in'")

for key in crew\_details:

    if(key=="Pilot" or key=="Co-Pilot"):

        print(crew\_details[key])

#Note: Dictionary being unordered, the order of the values being displayed may vary during each execution of the above for loop.

#Dictionaries are mutable

crew\_details["Pilot"]="James" # Here the value for key "Pilot" is being updated to "James"

print("\nAfter modifying the value of Pilot:", crew\_details["Pilot"])

print("------------------------------------------------------------------")

print("Before update:")

# Usage of get method()

print("Co-pilot:",crew\_details.get("Co-Pilot"))

#Usage of update method()

crew\_details.update({"Flight Attendant":"Jane", "Co-pilot":"Henry"})

print("\nAfter update:")

print("Co-pilot:",crew\_details.get("Co-pilot"))

print("Flight Attendant:",crew\_details["Flight Attendant"])

Iterating the dictionary using items function

Pilot : Kumar

Co-Pilot : Raghav

Head-Strewardess : Malini

Stewardess : Mala

Iterating the dictionary using keyword 'in'

Kumar

Raghav

After modifying the value of Pilot: James

------------------------------------------------------------------

Before update:

Co-pilot: Raghav

After update:

Co-pilot: Henry

Flight Attendant: Jane

Quiz – Dictionary

### **Q1 of 2**

What is the output of the following code snippet?

sample\_dict = {'a':1,'b':2}

sample\_dict.update({'b':5, 'c':10 })

print(sample\_dict.get('b'), sample\_dict.get('c'))

5,10

### **Q2 of 2**

Assume that a dictionary has data in the form of { key1:value1, key2:value2…}

my\_library =

{

103 : "Alice in Wonderland",

104 : "The Turning Point",

113 : "Wings on Fire",

134 : "Harry Potter"

}

What do you think my\_library[104] would point to ?

The Turning Point

Exercise on Dictionary - Level 1

'''

Problem Statement

Represent a small bilingual (English-Swedish) glossary given below as a Python dictionary

{"merry":"god", "christmas":"jul", "and":"och", "happy":"gott", "new":"nytt", "year":"ar"}

and use it to translate your Christmas wishes from English into Swedish.

That is, write a python function translate() that accepts the bilingual dictionary and a list of English words (your Christmas wish) and returns a list of equivalent Swedish words.

'''

def translate(bilingual\_dict,english\_words\_list):

    swedish\_words\_list=[]

    for i in english\_words\_list:

        if i in bilingual\_dict.keys():

            swedish\_words\_list.append(bilingual\_dict[i])

    return swedish\_words\_list

bilingual\_dict= {"merry":"god", "christmas":"jul", "and":"och", "happy":"gott", "new":"nytt", "year":"ar"}

english\_words\_list=["merry","christmas"]

print("The bilingual dict is:",bilingual\_dict)

print("The english words are:",english\_words\_list)

swedish\_words\_list=translate(bilingual\_dict, english\_words\_list)

print("The equivalent swedish words are:",swedish\_words\_list)

The bilingual dict is: {'merry': 'god', 'christmas': 'jul', 'and': 'och', 'happy': 'gott', 'new': 'nytt', 'year': 'ar'}

The english words are: ['merry', 'christmas']

The equivalent swedish words are: ['god', 'jul']

Assignment on String - Level 2

Given a string containing uppercase characters (A-Z), compress the string using Run Length encoding. Repetition of character has to be replaced by storing the length of that run.  
  
Write a python function which performs the run length encoding for a given String and returns the run length encoded String.  
  
Provide different String values and test your program

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| AAAABBBBCCCCCCCC | 4A4B8C |
| AABCCA | 2A1B2C1A |

'''

Given a string containing uppercase characters (A-Z), compress the string using Run Length encoding. Repetition of character has to be replaced by storing the length of that run.

Write a python function which performs the run length encoding for a given String and returns the run length encoded String.

'''

def encode(message):

    counter = 1

    result = ""

    previousLetter = message[0]

    if len(message)==1:

      return str(1) + message[0]

    for i in range(1,len(message),1):

        if not message[i] == previousLetter:

            result += str(counter) + message[i-1]

            previousLetter = message[i]

            counter = 1

        else:

            counter += 1

        if i == len(message)-1:

                result += str(counter) + message[i]

    return result

#Provide different values for message and test your program

encoded\_message=encode("ABBBBCCCCCCCCAB")

print(encoded\_message)

Assignment on Dictionary - Level 3

Care hospital wants to know the medical speciality visited by the maximum number of patients. Assume that the patient id of the patient along with the medical speciality visited by the patient is stored in a list. The details of the medical specialities are stored in a dictionary as follows:  
{  
"P":"Pediatrics",  
"O":"Orthopedics",  
"E":"ENT  
}   
  
Write a function to find the medical speciality visited by the maximum number of patients and return the name of the speciality.  
  
**Note:**

1. Assume that there is always only one medical speciality which is visited by maximum number of patients.
2. Perform case sensitive string comparison wherever necessary.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| [101,P,102,O,302,P,305,P] | Pediatrics |
| [101,O,102,O,302,P,305,E,401,O,656,O] | Orthopedics |
| [101,O,102,E,302,P,305,P,401,E,656,O,987,E] | ENT |

'''

Care hospital wants to know the medical speciality visited by the maximum number of patients. Assume that the patient id of the patient along with the medical speciality visited by the patient is stored in a list. The details of the medical specialities are stored in a dictionary as follows:

{

"P":"Pediatrics",

"O":"Orthopedics",

"E":"ENT

}

Write a function to find the medical speciality visited by the maximum number of patients and return the name of the speciality.

Note:

Assume that there is always only one medical speciality which is visited by maximum number of patients.

a\_dictionary = {"a": 1, "b": 2, "c": 3}

max\_key = max(a\_dictionary, key=a\_dictionary.get)   get key with max value

print(max\_key)

all\_values = a\_dictionary.values()

max\_value = max(all\_values)     all\_values is a list

print(max\_value)

'''

def max\_visited\_speciality(patient\_medical\_speciality\_list,medical\_speciality):

    patient\_medical\_speciality\_list= Convert(patient\_medical\_speciality\_list)

    # print(patient\_medical\_speciality\_list)

    s\_list={}

    for i in medical\_speciality:

        s\_list[i]=0

    for key,val in patient\_medical\_speciality\_list.items():

        s\_list[val]+=key

    speciality=medical\_speciality[(max(s\_list,key=s\_list.get))]

    print(s\_list)

    return speciality

# def Convert(lst):

#     res\_dct = {lst[i]: lst[i + 1] for i in range(0, len(lst), 2)}

#     return res\_dct

def Convert(lst):

    it = iter(lst)

    res\_dct = dict(zip(it, it))

    return res\_dct

#provide different values in the list and test your program

# patient\_medical\_speciality\_list=[301,'P',302, 'P' ,305, 'P' ,401, 'E' ,1656, 'E']

# medical\_speciality={"P":"Pediatrics","O":"Orthopedics","E":"ENT"}

patient\_medical\_speciality\_list=[101, 'O', 102, 'O', 302, 'P', 305, 'E', 401, 'O', 656,'P']

medical\_speciality={'O': 'Orthopedics', 'P': 'Pediatrics', 'E': 'ENT'}

speciality=max\_visited\_speciality(patient\_medical\_speciality\_list,medical\_speciality)

print(speciality)

Assignment on String - Level 1

### **Problem Statement**

Write a function, check\_palindrome() to check whether the given string is a palindrome or not. The function should return true if it is a palindrome else it should return false.  
  
Note: Initialize the string with various values and test your program. Assume that all the letters in the given string are all of the same case. Example: MAN, civic, WOW etc.

Argument Behavior

'''

Immutable- as value (roughly said) integer tuple string we pass copy

Mutabble - as refereence list dictionary we pass reference

'''

#integer

# def change\_no(num):

#     num+=10

# num\_val=20

# change\_no(num\_val)

# print(num\_val)

#string

# def change\_no(num):

#     num+="word"

#     print("Inside",num)

# num\_val="sample"

# change\_no(num\_val)

# print("Outside",num\_val)

# list

def change\_no(num):

    num.append(3)

    print("Inside",num)

num\_val=[1,2]

change\_no(num\_val)

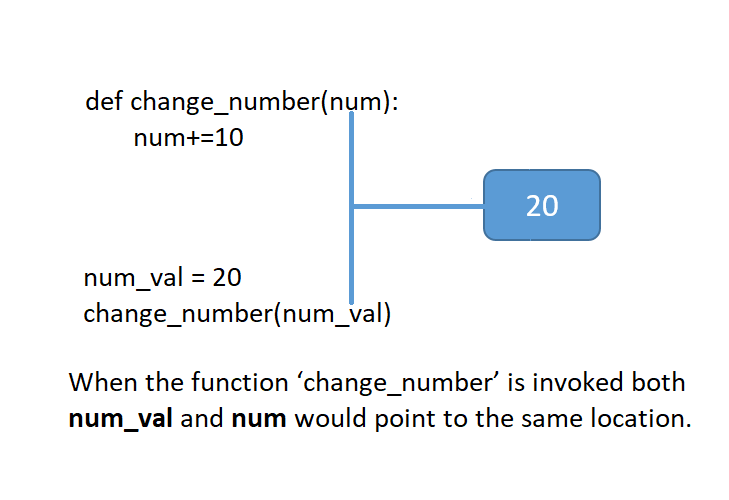
print("Outside",num\_val)

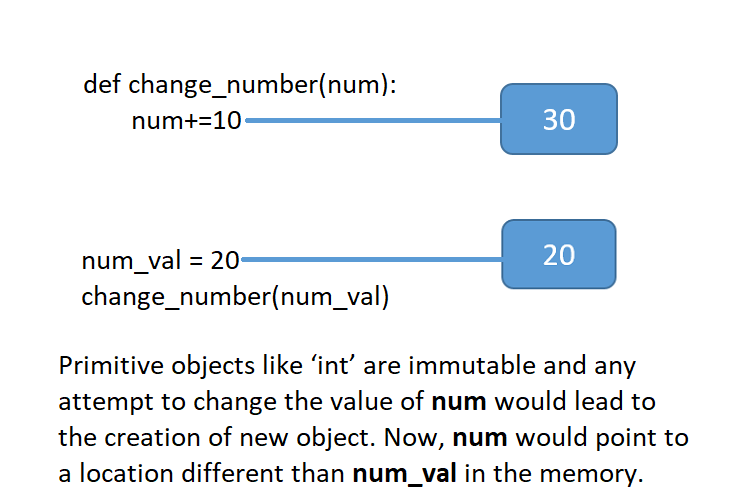
Argument to functions

In programming, there are two ways in which arguments can be passed to functions: **pass by value** and **pass by reference.**  
Some languages use **pass by value** by default while others use **pass by reference**. Some languages support both and allow you to choose.

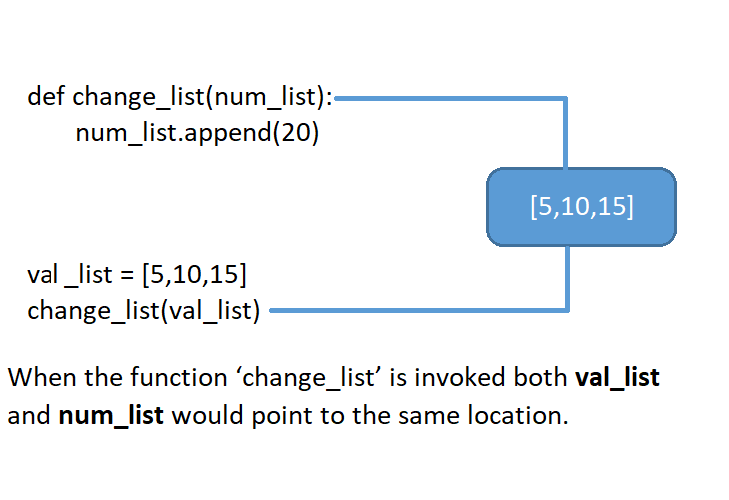
Everything in Python is considered to be an object and can be categorized as either ‘Mutable’ or ‘Immutable’. In Python, arguments are passed by reference.

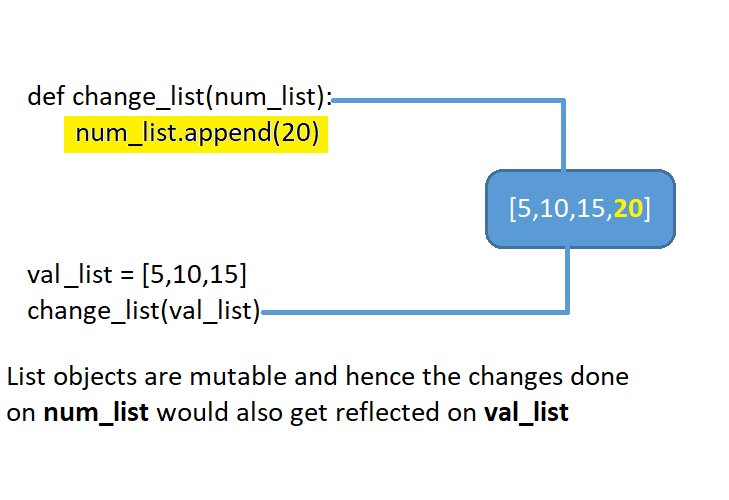
**Pass by reference - Immutable Objects**

****



**Pass by reference – Mutable Objects**

****

****

Try out - Argument Behavior

def change\_number(num):

    num+=10

def change\_list(num\_list):

    num\_list.append(20)

num\_val=10

print("num\_val before function call:", num\_val)

change\_number(num\_val)

print("num\_val after function call:", num\_val)

print("-----------------------------------------------")

val\_list=[5,10,15]

print("val\_list before function call:", val\_list)

change\_list(val\_list)

print("val\_list after function call:", val\_list)

num\_val before function call: 10

num\_val after function call: 10

-----------------------------------------------

val\_list before function call: [5, 10, 15]

val\_list after function call: [5, 10, 15, 20]

Try out - Arguments to Functions

In Python, when an argument passed to a function is modified inside it, whether the change is visible outside or not depends on whether the variable is mutable or not.

Let's understand how it works. Execute and observe the result for code given below;

def check\_in(baggage,boarding\_pass):

    if(baggage>=1 and baggage<=30):

            boarding\_pass="Issued"

def update\_seat(seat\_list):

    seat\_list[1]=25

boarding\_pass="Not Issued"

print("boarding\_pass before function call:", boarding\_pass)

check\_in(25, boarding\_pass)

print("boarding\_pass after function call:", boarding\_pass)

print("boarding\_pass, a string is immutable")

print("-------------------------------------------------------")

passenger\_seat=["Jack","NA"]

print("passenger\_seat before function call:", passenger\_seat)

update\_seat(passenger\_seat)

print("passenger\_seat after function call:", passenger\_seat)

print("passenger\_seat, a list is mutable")

boarding\_pass before function call: Not Issued

boarding\_pass after function call: Not Issued

boarding\_pass, a string is immutable

-------------------------------------------------------

passenger\_seat before function call: ['Jack', 'NA']

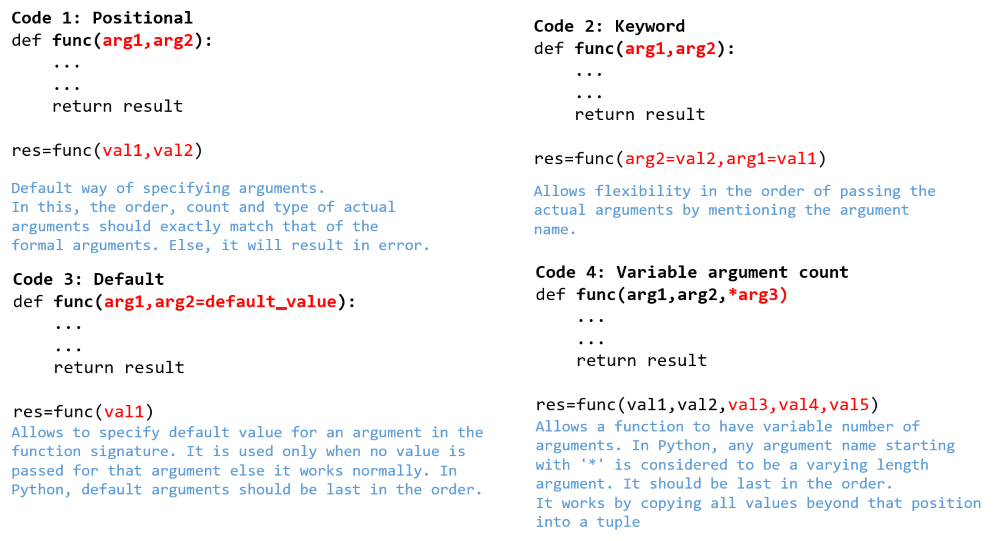
passenger\_seat after function call: ['Jack', 25]

passenger\_seat, a list is mutable

Types of Arguments

Function arguments - Ordering and default values

Programming languages allow controlling the ordering and default values of arguments.  
In python we will observe the following:



Try out - Types of Arguments

def display1(flight\_number, seating\_capacity):

    print("Flight Number:", flight\_number)

    print("Seating Capacity:", seating\_capacity)

print("code-1: positional arguments")

display1("AI789",200)

#Uncomment and execute the below function call statement and observe the output

#display1(300,"BA123")

def display2(flight\_number, seating\_capacity):

    print("Flight Number:", flight\_number)

    print("Seating Capacity:", seating\_capacity)

print("-------------------------------------------------")

print("code-2: keyword arguments")

display2(seating\_capacity=250, flight\_number="AI789")

def display3(flight\_number, flight\_make="Boeing", seating\_capacity=150):

    print("Flight Number:", flight\_number)

    print("Flight Make:", flight\_make)

    print("Seating Capacity:", seating\_capacity)

print("-------------------------------------------------")

print("code-3: default arguments")

display3("AI789","Eagle")

#Uncomment and execute the below function call statements one by one and observe the output

#display3("BA234")

#display3("SI678","Qantas",200)

def display4(passenger\_name, \*baggage\_tuple):

    print("Passenger name:",passenger\_name)

    total\_wt=0

    for baggage\_wt in baggage\_tuple:

        total\_wt+=baggage\_wt

    print("Total baggage weight in kg:", total\_wt)

print("-------------------------------------------------")

print("code-4: variable argument count")

display4("Jack",12,8,5)

#Uncomment and execute the below function call statements one by one and observe the output

#display4("Chan",20,12)

#display4("Henry",23)

code-1: positional arguments

Flight Number: AI789

Seating Capacity: 200

-------------------------------------------------

code-2: keyword arguments

Flight Number: AI789

Seating Capacity: 250

-------------------------------------------------

code-3: default arguments

Flight Number: AI789

Flight Make: Eagle

Seating Capacity: 150

-------------------------------------------------

code-4: variable argument count

Passenger name: Jack

Total baggage weight in kg: 25

Default argument defined at last

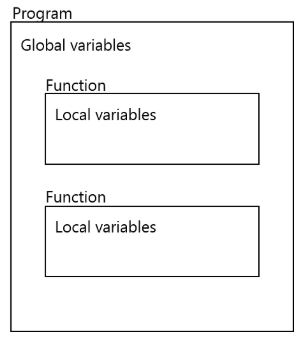
Positional argument has be defined before keyword argument

Types of Variables

There are two types of variables which we can use in a program - local variables and global variables.

Local variables are the ones which are created inside a function. They are created when the owning function starts execution and remains in memory till owning function finishes execution. They can be accessed only inside that function.

Global variables are the ones which are created outside the functions. They are created when the program execution starts and remains in memory till the program terminates. They can be read anywhere in the program - within a function or outside.



Try out - Variables & its Scope

The below code has been written to represent the baggage weight check process based on the weight limit specified by an airline.  
Go through the below code and guess the output.

wt\_limit=30

def baggage\_check(baggage\_wt):

extra\_baggage\_charge=0

if not(baggage\_wt>=0 and baggage\_wt<=wt\_limit):

extra\_baggage=baggage\_wt-wt\_limit

extra\_baggage\_charge=extra\_baggage\*100

return extra\_baggage\_charge

def update\_baggage\_limit(new\_wt\_limit):

wt\_limit=new\_wt\_limit

print("This airline now allows baggage limit till",wt\_limit,"kgs")

print("This airline allows baggage limit till",wt\_limit,"kgs")

print("Pay the extra baggage charge of",baggage\_check(35),"rupees")

update\_baggage\_limit(45)

print("Pay the extra baggage charge of",baggage\_check(35),"rupees")

wt\_limit=30

def baggage\_check(baggage\_wt):

    extra\_baggage\_charge=0

    if not(baggage\_wt>=0 and baggage\_wt<=wt\_limit):

        extra\_baggage=baggage\_wt-wt\_limit

        extra\_baggage\_charge=extra\_baggage\*100

    return extra\_baggage\_charge

def update\_baggage\_limit(new\_wt\_limit):

    wt\_limit=new\_wt\_limit

    print("This airline now allows baggage limit till",wt\_limit,"kgs")

print("This airline allows baggage limit till",wt\_limit,"kgs")

print("Pay the extra baggage charge of",baggage\_check(35),"rupees")

update\_baggage\_limit(45)

print("Pay the extra baggage charge of",baggage\_check(35),"rupees")

This airline allows baggage limit till 30 kgs

Pay the extra baggage charge of 500 rupees

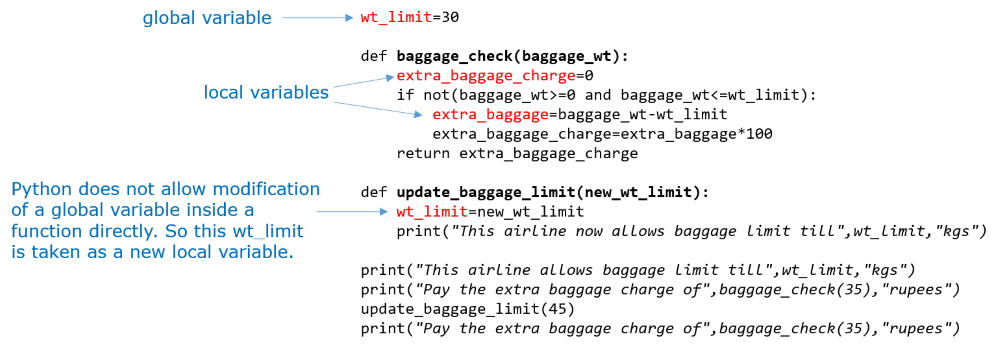
This airline now allows baggage limit till 45 kgs

Pay the extra baggage charge of 500 rupees

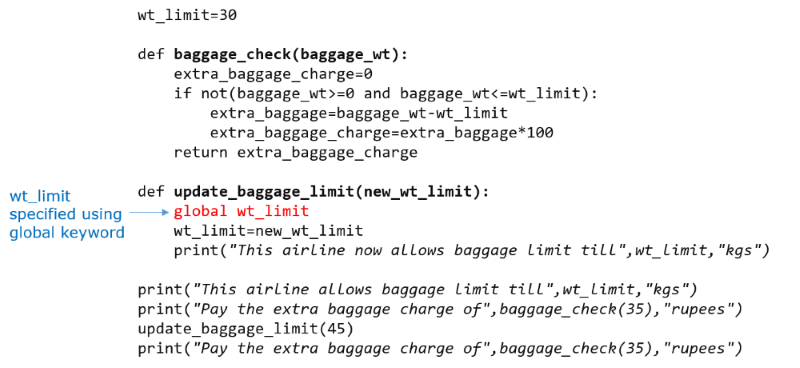
Variables & its Scope

**extra\_baggage** and **extra\_baggage\_charge** are created inside the function baggage\_check(). Hence they are local to that function or in other words, they are **local variables**. They are created when owning function starts execution and remains in memory till owning function finishes execution. They can be accessed only inside that function.

**wt\_limit** is created outside the functions. Hence it is a **global variable**. Global variables are created when the program execution starts and remains in memory till the program terminates. They can be read anywhere in the program - within a function or outside. But they are protected from modification inside a function. As it is available throughout the program, use of global variable should be restricted to avoid accidental misuse by developers and to minimize memory usage.



In cases where a global variable needs to be modified inside a function, like in function update\_baggage\_limit(), Python allows you to do that using the global keyword.



Try out - 'global'

**Part A: Make the code executable:**

* Fix the code to observe local variable cannot be accessed anywhere outside the owning function (you can comment code without losing functionality)

**Part B: Execute and observe the following 4 points**

* keyword global which allows to modify global variable wt\_limit inside update\_baggage\_limit() function
* global variable created in the beginning and remains till end
* global variable can be accessed within any function as well as outside a function
* local variable is created when func. starts, removed when func. completes

wt\_limit=30

def baggage\_check(baggage\_wt):

    extra\_baggage\_charge=0

    if not(baggage\_wt>=0 and baggage\_wt<=wt\_limit):

        extra\_baggage=baggage\_wt-wt\_limit

        extra\_baggage\_charge=extra\_baggage\*100

    return extra\_baggage\_charge

def update\_baggage\_limit(new\_wt\_limit):

    global wt\_limit

    wt\_limit=new\_wt\_limit

    print("This airline now allows baggage limit till",wt\_limit,"kgs")

def useless\_function\_to\_prove\_a\_point():

    print("Extra baggage:",extra\_baggage)

    print("Extra baggage charge:",extra\_baggage\_charge)

print("This airline allows baggage limit till",wt\_limit,"kgs")

print("Pay the extra baggage charge of",baggage\_check(35),"rupees")

print("Extra baggage:",extra\_baggage)

print("Extra baggage charge:",extra\_baggage\_charge)

update\_baggage\_limit(45)

print("Pay the extra baggage charge of",baggage\_check(35),"rupees")

useless\_function\_to\_prove\_a\_point()

This airline allows baggage limit till 30 kgs

Pay the extra baggage charge of 500 rupees

Traceback (most recent call last):

  File "d:\STUDY\INFY\GENERIC\PROGRAM\tempCodeRunnerFile.py", line 21, in <module>

    print("Extra baggage:",extra\_baggage)

NameError: name 'extra\_baggage' is not defined

Quiz - Variables & its Scope

### **Q1 of 6**

What will be the value of the variables total and game\_points\_history, after executing the below code?

def check\_loss(game\_history,game\_points,total):

if game\_history[1] == 0:

game\_points[1] -= 1

else:

loss\_points = game\_history[1] \* 2

game\_points[1] -= loss\_points

total = game\_points[0] + game\_points[1] + game\_points[2]

game\_history = [4,0,2]

game\_points = [12,-4,2]

total = 0

check\_loss(game\_history, game\_points, total)

game\_history = [5,2,2]

check\_loss(game\_history, game\_points, total)

print(total,game\_points)

0, [12, -9, 2]

### **Q2 of 6**

Predict the output of the following code snippet.

result=0

def find\_sum(num1,num2):

    if(num1!=num2):

        result=num1+num2

    else:

        result=2\*(num1+num2)

find\_sum(3,4)

print(result)

find\_sum(5,5)

print(result)

a. 0  
    0

b. 14  
    10

c. 7  
   20

d. Error:cannot modify global variable in a program

a

### **Q3 of 6**

What is the output of the following code snippet?

def find\_avg(list\_num):

    result\_sum=0

    for num in list\_num:

        result\_sum+=num

    result\_avg=result\_sum/len(list\_num)

find\_avg([5,8,5])

print(result\_avg)

Error: 'result\_avg' is not defined

### **Q4 of 6**

What is the output of the following code snippet?

def func(word, char="A"):

    if(char=="A"):

        return len(word[1:])

    elif(char=="B"):

        return len(word[2:])

    else:

        return len(word)

print(func("Apple","A"),end=" ")

print(func("Apple","B"),end=" ")

print(func("Apple"),end=" ")

print(func("Apple","C"),end=" ")

4 3 4 5

### **Q5 of 6**

Choose the statements which are CORRECT with respect to the below code

1. arg2 is a default argument
2. We cannot have a positional argument after arg2 in func1
3. arg4 is a variable length argument
4. We can have any number of default arguments after arg4 in func2
5. arg5 and arg6 are positional arguments
6. arg1 and arg3 are positional arguments
7. \*argument\_name indicates default argument

def func1(arg1,arg2 = 5):

if(arg1 > arg2):

return arg1

return arg2

def func2(arg3,\*arg4):

for i in arg4:

if(arg3 == i):

return i

return 0

def func3(arg5,arg6):

if(arg5 == arg6):

return True

return False

res1 = func1(1)

res2 = func2(res1,1,1,2,5,7,8)

print(func3(arg6 = 5,arg5 = res2))

1,2,3,6

### **Q6 of 6**

Predict the output of the following code snippet.

def func(sample, res, key,val):

index =- 1

if(key in sample):

res = True

index = sample.index(key)

values[index] = val

else:

res = False

return index

res = None

sample = ["A","B","C","D"]

values = [1,1,3,4,5]

index = func(sample,res,"B",2)

print(values[index], res)

2 None

Exercise on Function Arguments - Level 3

'''Write a Python function to find all the Strong numbers in a given list of numbers.

Write another function to find and return the factorial of a number. Use it to solve the problem.

Example:

A number is considered to be a Strong number if sum of the factorial of its digits is equal to the number itself.

145 is a Strong number as 1! + 4! + 5! = 145.

Sample Input

num\_list = [145,375,0,100,2]

Expected Output

[145, 2]

Note: 0!=1

'''

def factorial(number):

    if number==0:

        return 1

    elif number==1:

        return number

    else:

        return number\*factorial(number-1)

def find\_strong\_numbers(num\_list):

    s\_no\_list=[]

    for i in num\_list:

        temp=i

        sum=0

        # print(temp)

        while temp>0:

            rem=temp%10

            # print(rem)

            sum+=factorial(rem)

            temp//=10

        # print(sum)

        if i==sum:

            s\_no\_list.append(i)

    return s\_no\_list

num\_list=[145,375,100,2,10]

strong\_num\_list=find\_strong\_numbers(num\_list)

print(strong\_num\_list)

Try out - Exception Handling - The Scenario

#calculating airport expenditure

def calculate\_expenditure(list\_of\_expenditure):

    total=0

    for expenditure in list\_of\_expenditure:

        total+=expenditure

    print(total)

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

  File "d:\STUDY\INFY\GENERIC\PROGRAM\TO\_Exception\_handling.py", line 8, in <module>

    calculate\_expenditure(list\_of\_values)

  File "d:\STUDY\INFY\GENERIC\PROGRAM\TO\_Exception\_handling.py", line 5, in calculate\_expenditure

    total+=expenditure

TypeError: unsupported operand type(s) for +=: 'int' and 'str'

### **Problem Statement**

We got an error in the below code, one way to take care of such error situation is to use selection constructs.  
The error was due to addition of a string (“400”) to an integer. If we add a condition to check whether the expenditure is of type int, that would solve this error. But that can cause further issues.

#calculating airport expenditure

def calculate\_expenditure(list\_of\_expenditure):

total=0

for expenditure in list\_of\_expenditure:

total+=expenditure

print(total)

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

Execute the below code and observe the results.

Try out - Exception Handling using 'if'

#calculating airport expenditure

def calculate\_expenditure(list\_of\_expenditure):

    total=0

    for expenditure in list\_of\_expenditure:

        if(type(expenditure) is int):

            total+=expenditure

        else:

            print("Wrong data type")

            break

    print(total)

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

Wrong data type

600

Try out - Exception Handling using 'try' & 'except'

In python we can create a try and except block of code.  
If any error occurs in the try block of code, it will jump to except block of code.  
Once the except block is executed, the code continues to execute other statements outside except block.

#calculating airport expenditure

def calculate\_expenditure(list\_of\_expenditure):

    total=0

    try:

        for expenditure in list\_of\_expenditure:

            total+=expenditure

        print(total)

    except:

        print("Some error occured")

    print("Returning back from function.")

list\_of\_values=[100,200,300,"400",500]

calculate\_expenditure(list\_of\_values)

Some error occured

Returning back from function.

Exception Handling - Different Types

Python has many kinds of errors predefined as part of the language. Here are some of the common types.

|  |  |  |
| --- | --- | --- |
| **Built-in Exception** | **When it will be raised** | **Example** |
| ZeroDivisionError | When a value is divided by zero | num\_list=[] total=10 avg=total/len(num\_list) |
| TypeError | When we try to do an operation with incompatible data types | total=10 total+="20" |
| NameError | When we try to access a variable which is not defined | avg=total/10 where total is not defined |
| IndexError | When we try to access an index value which is out of range | num\_list=[1,2,3,4] value=num\_list[4] |
| ValueError | When we use a valid data type for an argument of a built-in function but passes an invalid value for it | #string is a valid data type for int() but the value “A” is invalid, as "A" can't be converted into int. value="A" num=int(value) |

 Python also allows us to handle different errors that can occur separately. That means you can have a different action or message for every unique error that occurs.

Here is the same expenditure calculation code with additional average expenditure calculation.

def calculate\_expenditure(list\_of\_expenditure):

    total=0

    try:

        for expenditure in list\_of\_expenditure:

            total+=expenditure

        print("Total:",total)

        avg=total/num\_values

        print("Average:",avg)

    except ZeroDivisionError:

        print("Divide by Zero error")

    except TypeError:

        print("Wrong data type")

    except:

        print("Some error occured")

list\_of\_values=[100,200,300,"400",500]

num\_values=0

calculate\_expenditure(list\_of\_values)

def calculate\_expenditure(list\_of\_expenditure):

    total=0

    try:

        for expenditure in list\_of\_expenditure:

            total+=expenditure

        print("Total:",total)

        avg=total/num\_values

        print("Average:",avg)

    except ZeroDivisionError:

        print("Divide by Zero error")

    except TypeError:

        print("Wrong data type")

    except:

        print("Some error occured")

list\_of\_values=[100,200,300,"400",500]

num\_values=0

calculate\_expenditure(list\_of\_values)

**Note:**

1. Default except block is the one without any type mentioned.
2. If an error occurs and the matching except block is found, then that is executed.
3. If an error occurs and the matching except block is not found, it executes the default except block.
4. If an error occurs and the matching except block is not found and if the default except block is also not found, the code crashes.
5. The default except block, if present should be the last except block, otherwise it will result in a runtime error.

Try out - Customized Error Messages

def calculate\_expenditure(list\_of\_expenditure):

    total=0

    try:

        for expenditure in list\_of\_expenditure:

            total+=expenditure

        print("Total:",total)

        avg=total/num\_values

        print("Average:",avg)

    except:

        print("Some error occured")

    except ZeroDivisionError:

        print("Divide by Zero error")

    except TypeError:

        print("Wrong data type")

list\_of\_values=[100,200,300,"400",500]

num\_values=0

calculate\_expenditure(list\_of\_values)

  File "d:\STUDY\INFY\GENERIC\PROGRAM\tempCodeRunnerFile.py", line 8

    print("Average:",avg)

                    ^

SyntaxError: default 'except:' must be last

Try out - Multiple Try Blocks

If an error occurs inside a function and if the error is not caught inside it, then the error is transferred to the function call where we have another opportunity to catch it.

Execute the below code and observe the results.

def calculate\_sum(list\_of\_expenditure):

    total=0

    try:

        for expenditure in list\_of\_expenditure:

            total+=expenditure

        print("Total:",total)

        avg=total/no\_values

        print("Average:",avg)

    except ZeroDivisionError:

        print("Divide by Zero error")

    except TypeError:

        print("Wrong data type")

try:

    list\_of\_values=[100,200,300,400,500]

    num\_values=len(list\_of\_values)

    calculate\_sum(list\_of\_values)

except NameError:

    print("Name error occured")

except:

    print("Some error occured")

Total: 1500

Name error occured

Try out - Finally Block

### **Problem Statement**

Sometimes in programming we need to execute some code irrespective of whether the primary program logic itself succeeds or fails to do its job. In Python we can achieve this using a **finally** block.

A **finally** block of statement is an optional part of the try-except statements. A code written inside the finally block will ALWAYS be executed.

Execute the below code and observe the results.

balance=1000

amount="300Rs"

def take\_card():

    print("Take the card out of ATM")

try:

    if balance>=int(amount):

        print("Withdraw")

    else:

        print("Invalid amount")

except TypeError:

    print("Type Error Occurred")

except ValueError:

    print("Value Error Occurred")

except:

    print("Some error Occurred")

finally:

    take\_card()

balance=1000

amount="300Rs"

def take\_card():

    print("Take the card out of ATM")

try:

    if balance>=int(amount):

        print("Withdraw")

    else:

        print("Invalid amount")

except TypeError:

    print("Type Error Occurred")

except ValueError:

    print("Value Error Occurred")

except:

    print("Some error Occurred")

finally:

    take\_card()

Value Error Occurred

Take the card out of ATM

Value Error Occurred

Take the card out of ATM

Quiz - Exception Handling

### **Q1 of 4**

### What will be the output of the below code?

def divide(num1, num2):

try:

num3 = num1 / num2

print(num3)

except:

print("Inside function")

try:

divide(100, 0)

except:

print("Outside function")

Inside function

### **Q2 of 4**

Go through the below code and predict the output:

num1=100

num2=0

try:

    result=num1/num2

    print(result)

except ZeroDivisionError:

    print("Zero Division Error Occurred")

Zero Division Error Occurred

### **Q3 of 4**

What will be the output of the code given below?

def division(value\_1,value\_2):

try:

return int(value\_1)/value\_2

except TypeError:

print("Type error")

except ValueError:

print("Value error")

finally:

print("Finally")

print("Done")

division('A',10)

a. Value error  
    Finally  
    Done

b. Type error  
    Finally  
    Done

c. Type error  
    Finally

d. Value error  
    Finally

a

### **Q4 of 4**

What will be the output of the below code?

def find\_sum(value\_1,value\_2):

try:

print(value\_1+value\_3)

except NameError:

print("Function name error")

finally:

print("Sum finally")

try:

find\_sum(12,13)

except NameError:

print("Invocation name error")

finally:

print("Invocation finally")

a. Function name error  
    Sum finally

b. Function name error  
    Sum finally  
    Invocation finally

c. Function name error  
    Sum finally  
    Invocation name error  
    Invocation finally

d. Sum finally  
    Invocation name error  
    Invocation finally

b

Exercise on Exception Handling - Level 1

### **Problem Statement**

The python function, **find\_average()** given below, is written to accept a list of marks and return the average marks. On execution, the program is resulting in an error.  
  
1: Add code to handle the exception occurring in the code.  
  
2: Make the necessary correction in the program to remove the error.  
  
3: Make the following changes in the code, execute, observe the results. Add code to handle the errors occurring in each case.  
  
Case – 1: Initialize m\_list as ["1",2,3,4]  
  
Case – 2: Initialize m\_list as given below

mark1="A"

mark1=int("A")

m\_list=[mark1,2,3,4]

Case – 3: Initialize m\_list as []  
  
Case – 4: Make the following change in the for loop statement

for i in range(0, len(mark\_list)+1):