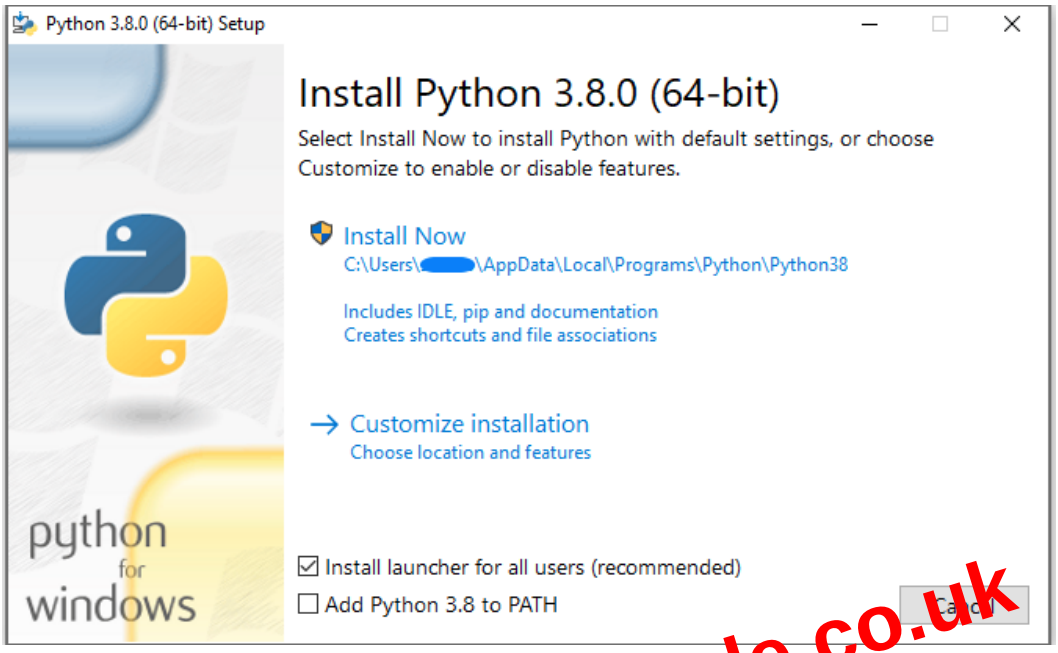
Step 5: Verify Pip Was Installed.

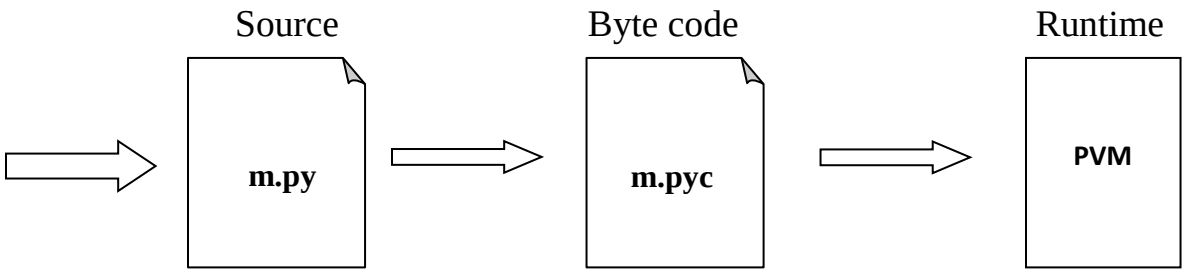
Step 6: Add Python Path to Environment Variables (Optional)



Working with Python

Python Code Execution:

Python's traditional runtime execution model: Source code you type is translated to byte code, which is then run by the Python Virtual Machine (PVM). Your code is automatically compiled, but then it is interpreted.



Source code extension is .py
Byte code extension is .pyc (Compiled python code)

There are two modes for using the Python interpreter:

- Interactive Mode
- Script Mode

Alternatively, programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file. To execute the script by the interpreter, you have to tell the interpreter the name of the file. For example, if you have a script name MyFile.py and you're working on Unix, to run the script you have to type:

python MyFile.py

Working with the interactive mode is better when Python programmers deal with small pieces of code as you can type and execute them immediately, but when the code is more than 2-4 lines, using the script for coding can help to modify and use the code in future.

Example:

```
C:\Users\MRCET\AppData\Local\Programs\Python\Python38-32\pyyy>python e1.py
resource open
the no cant be divisible zero division by zero
resource close
finished
```


Data types:

The data stored in memory can be of many types. For example, a student roll number is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Int:

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
>>> print(24656354687654+2)
24656354687656
>>> print(20)
20
>>> print(0b10)
2
```

```
>>> print('mrcet college')

mrcet college

>>> " "

' '
```

If you want to include either type of quote character within the string, the simplest way is to delimit the string with the other type. If a string is to contain a single quote, delimit it with double quotes and vice versa:

```
>>> print("mrcet is an autonomous (') college")

mrcet is an autonomous (') college

>>> print('mrcet is an autonomous (") college')

mrcet is an autonomous (") college
```

Suppressing Special Character:

Specifying a backslash (\) in front of the quote character in a string "escapes" it and causes Python to suppress its usual special meaning. It is then interpreted simply as a literal single quote character:


```
>>> print("mrcet is an autonomous (') college")

mrcet is an autonomous (') college

>>> print('mrcet is an autonomous (\") college')

mrcet is an autonomous (") college
```

The following is a table of escape sequences which cause Python to suppress the usual special interpretation of a character in a string:

```
>>> print('a\
....b')
a....b
>>> print('a\
b\
c')
```

UNIT – II

CONTROL FLOW, LOOPS

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: while, for, break, continue.

Control Flow, Loops:

Boolean Values and Operators:

A boolean expression is an expression that is either true or false. The following examples use the operator ==, which compares two operands and produces True if they are equal and False otherwise:

```
>>> 5 == 5
```

True

```
>>> 5 == 6
```

False

True and False are special values that belong to the type bool; they are not strings:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

The == operator is one of the relational operators; the others are: x != y # x is not equal to y

x > y # x is greater than y x < y # x is less than y

x >= y # x is greater than or equal to y x <= y # x is less than or equal to y

Note:

All expressions involving relational and logical operators will evaluate to either true or false

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/if1.py
3 is greater
done
-1 a is smaller
Finish
--------------------------------
```

```
a=10

if a>9:

   print("A is Greater than 9")
```

Output:

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/if2.py

A is Greater than 9
```

Alternative if (If-Else):

An else statement can be combined with an if statement. An else statement contains the block of code (false block) that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The else statement is an optional statement and there could be at most only one else Statement following if.

Syntax of if - else :

```
if test expression:
        Body of if stmts
else:
        Body of else stmts
```

If - else Flowchart :

```
j=1
while i<=3:
    print("MRCET",end="  ")

    while j<=1:
        print("CSE DEPT",end="")
        j=j+1
    i=i+1
    print()
```

Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/wh3.py

MRCET CSE DEPT
MRCET
MRCET

4. ----------

```
i = 1
while (i < 10):
      print (i)
      i = i+1
```

Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/wh4.py
1
2
3
4
5
6
7
8
9

2. ----------

```
a = 1
b = 1
while (a<10):
    print ('Iteration',a)
    a = a + 1
    b = b + 1
```

Iterating over a list:

```
#list of items
list = ['M','R','C','E','T']
i = 1

#Iterating over the list
for item in list:
  print ('college ',i,' is ',item)
  i = i+1
```

Output:

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/lis.py
college  1  is  M
college  2  is  R
college  3  is  C
college  4  is  E
college  5  is  T
```

Iterating over a Tuple:

```
tuple = (2,3,5,7)
print ('These are the first four prime numbers ')
#Iterating over the tuple
for a in tuple:
  print (a)
```

Output:

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/fr3.py
These are the first four prime numbers
2
3
5
7
```

Iterating over a dictionary:

```
#creating a dictionary
college = {"ces":"block1","it":"block2","ece":"block3"}

#Iterating over the dictionary to print keys
print ('Keys are:')
```

Terminating the loop

```
#-----------------------------------
for letter in "Python": # First Example
   if letter == "h":
      break
   print("Current Letter :", letter )
```

Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/br.py =

Current Letter : P

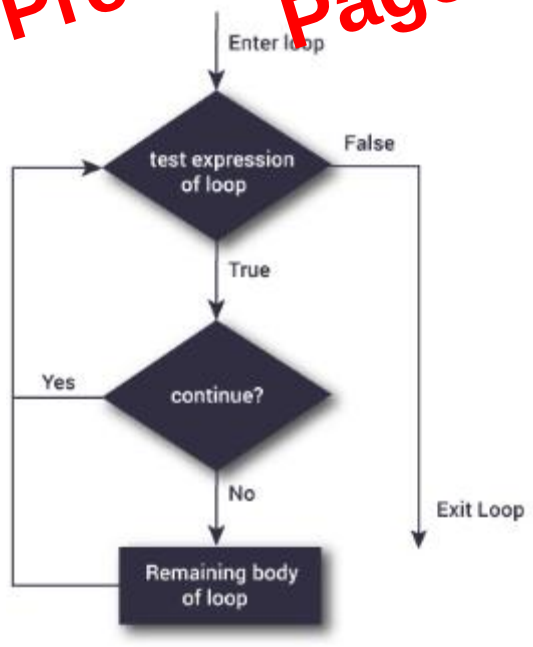
Current Letter : y

Current Letter : t

Continue:

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Flowchart:



The following shows the working of break statement in for and while loop:

True
True
None

```
#-----------------

def area(radius):

    b = 3.14159 * radius**2

    return b
```

Parameters:

Parameters are passed during the definition of function while Arguments are passed during the function call**.**

Example:

```
#here a and b are parameters

def add(a,b): #//function definition
 return a+b

#12 and 13 are arguments
#function call
result=add(12,13)
print(result)
```


Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/paraarg.py

25

Some examples on functions:

 # To display vandemataram by using function use no args no return type

```
#function defination
def display():
    print("vandemataram")
print("i am in main")
```

Isalnum() method returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

Syntax:
String.isalnum()

Example:
>>> string="123alpha"
>>> string.isalnum() True

2. isalpha():

isalpha() method returns true if string has at least 1 character and all characters are alphabetic and false otherwise.

Syntax:
String.isalpha()

Example:
>>> string="nikhil"
>>> string.isalpha()
True

3. isdigit():

isdigit() returns true if string contains only digits and false otherwise.

Syntax:
String.isdigit()

Example:
>>> string="123456789"
>>> string.isdigit()
True

4. islower():

Islower() returns true if string has characters that are in lowercase and false otherwise.

Syntax:

c	Represents character of size 1 byte
i	Represents signed integer of size 2 bytes
I	Represents unsigned integer of size 2 bytes
f	Represents floating point of size 4 bytes
d	Represents floating point of size 8 bytes

Creating an array:

```
from array import *
array1 = array('i', [10,20,30,40,50])
for x in array1:
 print(x)
```


Output:

```
>>>
 RESTART: C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/arr.py
10
20
30
40
50
```

Access the elements of an Array:

Accessing Array Element

We can access each element of an array using the index of the element.

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1[0])
print (array1[2])
```

UNIT – IV

LISTS, TUPLES, DICTIONARIES

Lists: list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters, list comprehension; Tuples: tuple assignment, tuple as return value, tuple comprehension; Dictionaries: operations and methods, comprehension;

Lists, Tuples, Dictionaries:

List:

- It is a general purpose most widely used in data structures
- List is a collection which is ordered and changeable and allows duplicate members. (Grow and shrink as needed, sequence type, sortable).
- To use a list, you must declare it first. Do this using square brackets and separate values with commas.
- We can construct / create list in many ways.

Ex:

```
>>> list1=[1,2,3,'A','B',7,8,[10,11]]
>>> print(list1)
[1, 2, 3, 'A', 'B', 7, 8, [10, 11]]
----------------------
>>> x=list()
>>> x
[]
-------------------------
>>> tuple1=(1,2,3,4)
>>> x=list(tuple1)
>>> x
[1, 2, 3, 4]
```

```
i = 1

#Iterating over the list
for item in list:
  print ('college ',i,' is ',item)
  i = i+1
```

Output:

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/lis.py
college  1  is  M
college  2  is  R
college  3  is  C
college  4  is  E
college  5  is  T
```

Method #2: For loop and range()

In case we want to use the traditional for loop which iterates from number x to number y.

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]

# getting length of list
length = len(list)

# Iterating the index
# same as 'for i in range(len(list))'
for i in range(length):
    print(list[i])
```


Output:

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/listlooop.py
1
3
5
7
9
```

Method #3: using while loop

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]

# Getting length of list
```

List comprehension:

List:

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

For example, assume we want to create a list of squares, like:

```
>>> list1=[]
>>> for x in range(10):
        list1.append(x**2)
>>> list1
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

(or)

This is also equivalent to

```
>>> list1=list(map(lambda x:x**2, range(10)))
>>> list1
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

(or)

Which is more concise and redable.

```
>>> list1=[x**2 for x in range(10)]
>>> list1
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Access tuple items: Access tuple items by referring to the index number, inside square brackets

```
>>> x=('a','b','c','g')
>>> print(x[2])
c
```

Change tuple items: Once a tuple is created, you cannot change its values. Tuples are unchangeable.

```
>>> x=(2,5,7,'4',8)
>>> x[1]=10
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    x[1]=10
TypeError: 'tuple' object does not support item assignment
>>> x
(2, 5, 7, '4', 8)     # the value is still the same
```

Loop through a tuple: We can loop the values of tuple using for loop

```
>>> x=4,5,6,7,2,'aa'
>>> for i in x:
        print(i)


4
5
6
7
2
aa
```


Count (): Returns the number of times a specified value occurs in a tuple

```
>>> x=(1,2,3,4,5,6,2,10,2,11,12,2)
>>> x.count(2)
4
```

Index (): Searches the tuple for a specified value and returns the position of where it was found

- The available option beside "w" are "r" for read and "a" for append and plus sign means if it is not there then create it

File Modes in Python:

Mode	Description
'r'	This is the default mode. It Opens file for reading.
'w'	This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
'x'	Creates a new file. If file already exists, the operation fails.
'a'	Open file in append mode. If file does not exist, it creates a new file.
't'	This is the default mode. It opens in text mode.
'b'	This opens in binary mode.
'+'	This will open a file for reading and writing (updating)

Reading and Writing files:

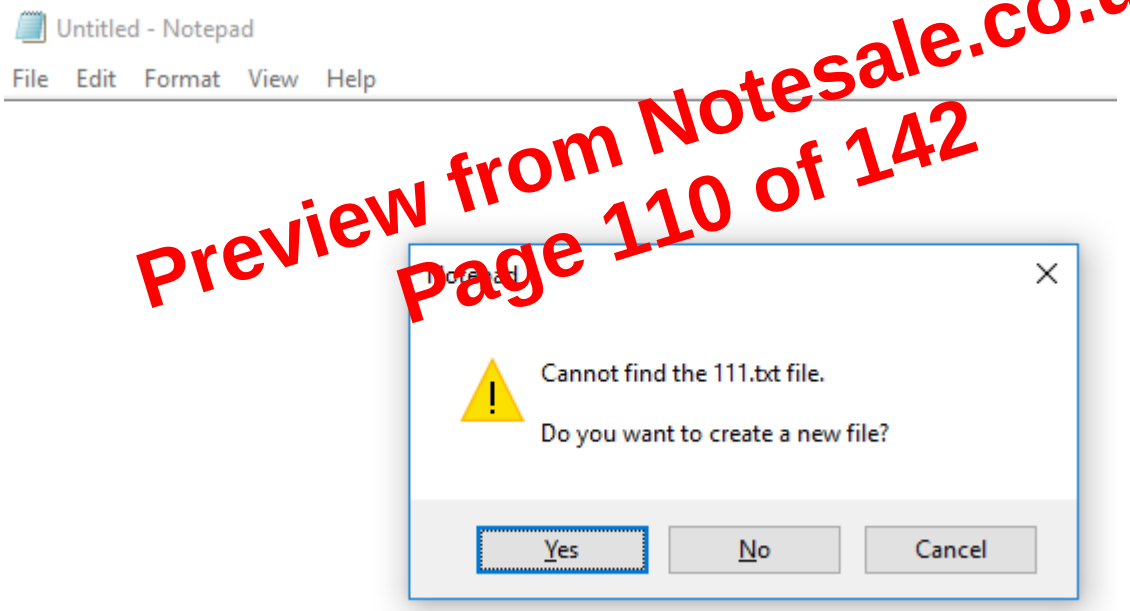
The following image shows how to create and open a text file in notepad from command prompt

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\MRCET\AppData\Local\Programs\Python\Python38-32\filess>start notepad hello.txt

C:\Users\MRCET\AppData\Local\Programs\Python\Python38-32\filess>type hello.txt
Hello mrcet
good morning
how r u
C:\Users\MRCET\AppData\Local\Programs\Python\Python38-32\filess>
```

(or)

```
C:\Users\MRCET\AppData\Local\Programs\Python\Python38-32\filess>notepad 111.txt
```

Hit on enter then it shows the following whether to open or not?



Click on “yes” to open else “no” to cancel

 # Write a python program to open and read a file

a=open(“one.txt”,”r”)

print(a.read())

Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/gtopt.py ==

['C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/gtopt.py']

Errors and Exceptions:

Python Errors and Built-in Exceptions: Python (interpreter) raises exceptions when it encounters **errors.** When writing a program, we, more often than not, will encounter errors. Error caused by not following the proper structure (syntax) of the language is called syntax error or parsing error

ZeroDivisionError:

ZeroDivisionError in Python indicates that the second argument used in a division (or modulo) operation was zero**.**

OverflowError:

OverflowError in Python indicates that an arithmetic operation has exceeded the limits of the current Python runtime. This is typically due to excessively large float values, as integer values that are too big will opt to raise memory errors instead.

ImportError:

It is raised when you try to import a module which does not exist. This may happen if you made a typing mistake in the module name or the module doesn't exist in its standard path. In the example below, a module named "non_existing_module" is being imported but it doesn't exist, hence an import error exception is raised.

IndexError:

An IndexError exception is raised when you refer a sequence which is out of range. In the example below, the list abc contains only 3 entries, but the 4th index is being accessed, which will result an IndexError exception.

TypeError:

When two unrelated type of objects are combined, TypeErrorexception is raised.In example below, an int and a string is added, which will result in TypeError exception.

IndentationError:

Unexpected indent. As mentioned in the "expected an indentedblock" section, Python not only insists on indentation, it insists on consistentindentation. You are free to choose the number of spaces of indentation to use, but you then need to stick with it.

Syntax errors:

These are the most basic type of error. They arise when the Python parser is unable to understand a line of code. Syntax errors are almost always fatal, i.e. there is almost never a way to successfully execute a piece of code containing syntax errors.

Run-time error:

A run-time error happens when Python understands what you are saying, but runs into trouble when following your instructions.

Key Error :

Python raises a KeyError whenever a dict() object is requested (using the format a = adict[key]) and the key is not in the dictionary.

Value Error:

In Python, a value is the information that is stored within a certain object. To encounter a ValueError in Python means that is a problem with the content of the object you tried to assign the value to.

Python has many built-in exceptions which forces your program to output an error when something in it goes wrong. In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class.

Different types of exceptions:

- ArrayIndexOutOfBoundException.
- ClassNotFoundException.
- FileNotFoundException.
- IOException.
- InterruptedException.
- NoSuchFieldException.
- NoSuchMethodException

Handling Exceptions:

The cause of an exception is often external to the program itself. For example, an incorrect input, a malfunctioning IO device etc. Because the program abruptly terminates on encountering an exception, it may cause damage to system resources, such as files. Hence, the exceptions should be properly handled so that an abrupt termination of the program is prevented.

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

Syntax:

```
try :
    #statements in try block
except :
    #executed when error in try block
```

Typically we see, most of the times

- **Syntactical errors** (wrong spelling, colon (:) missing ….),
  At developer level and compile level it gives errors.

- **Logical errors** (2+2=4, instead if we get output as 3 i.e., wrong output …..,),
  As a developer we test the application, during that time logical error may obtained.

- **Run time error** (In this case, if the user doesn't know to give input, 5/6 is ok but if the user say 6 and 0 i.e.,6/0 (shows error a number cannot be divided by zero))
  This is not easy compared to the above two errors because it is not done by the system, it is (mistake) done by the user.

The things we need to observe are:

1. You should be able to understand the mistakes; the error might be done by user, DB connection or server.
2. Whenever there is an error execution should not stop.
   Ex: Banking Transaction
3. The aim is execution should not stop even though an error occurs.

```
try:
    print(a/b)
except Exception:
    print("number can not be divided by zero")
    print("bye")
```

Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/ex3.py

number can not be divided by zero

bye

- **The except block executes only when try block has an error, check it below**

```
a=5
b=2
try:
    print(a/b)
except Exception:
    print("number can not be divided by zero")
    print("bye")
```


Output:

C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/ex4.py

2.5

- **For example if you want to print the message like what is an error in a program then we use “e” which is the representation or object of an exception.**

```
a=5
b=0
try:
```

```
        print("resource open")
        print(a/b)
        k=int(input("enter a number"))
        print(k)
except ZeroDivisionError as e:
        print("the value can not be divided by zero",e)
finally:
        print("resource closed")
```

Output:

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/ex10.py
resource open
the value can not be divided by zero division by zero
resource closed
```

- **change the value of b to 2 for above program, you see the output like**

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/ex10.py
resource open
2.5
enter a number 6
6
resource closed
```


- **Instead give input as some character or string for above program, check the output**

```
C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/ex10.py
resource open
2.5
enter a number p
resource closed
Traceback (most recent call last):
  File "C:/Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/ex10.py", line 7, in <module>
    k=int(input("enter a number"))
ValueError: invalid literal for int() with base 10: ' p'
```