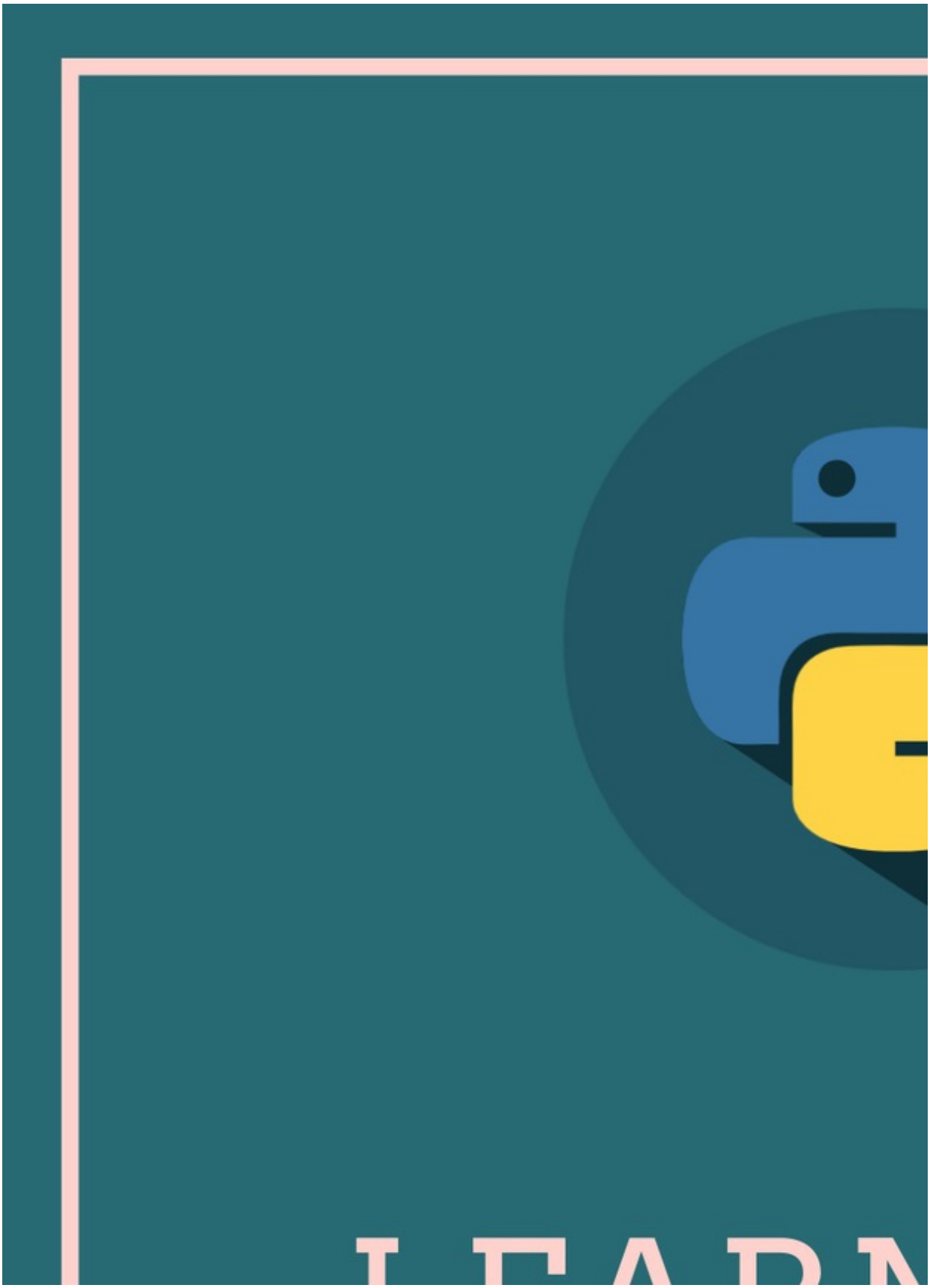
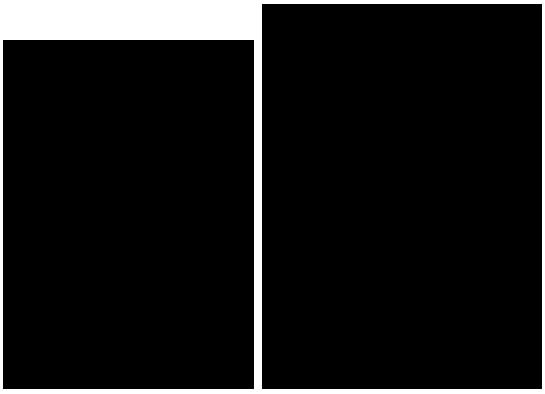
LEARN AND PRACTICE PYTHON IN 7 DAYS FROM ZERO TO HERO

BY OSAMA ADEL

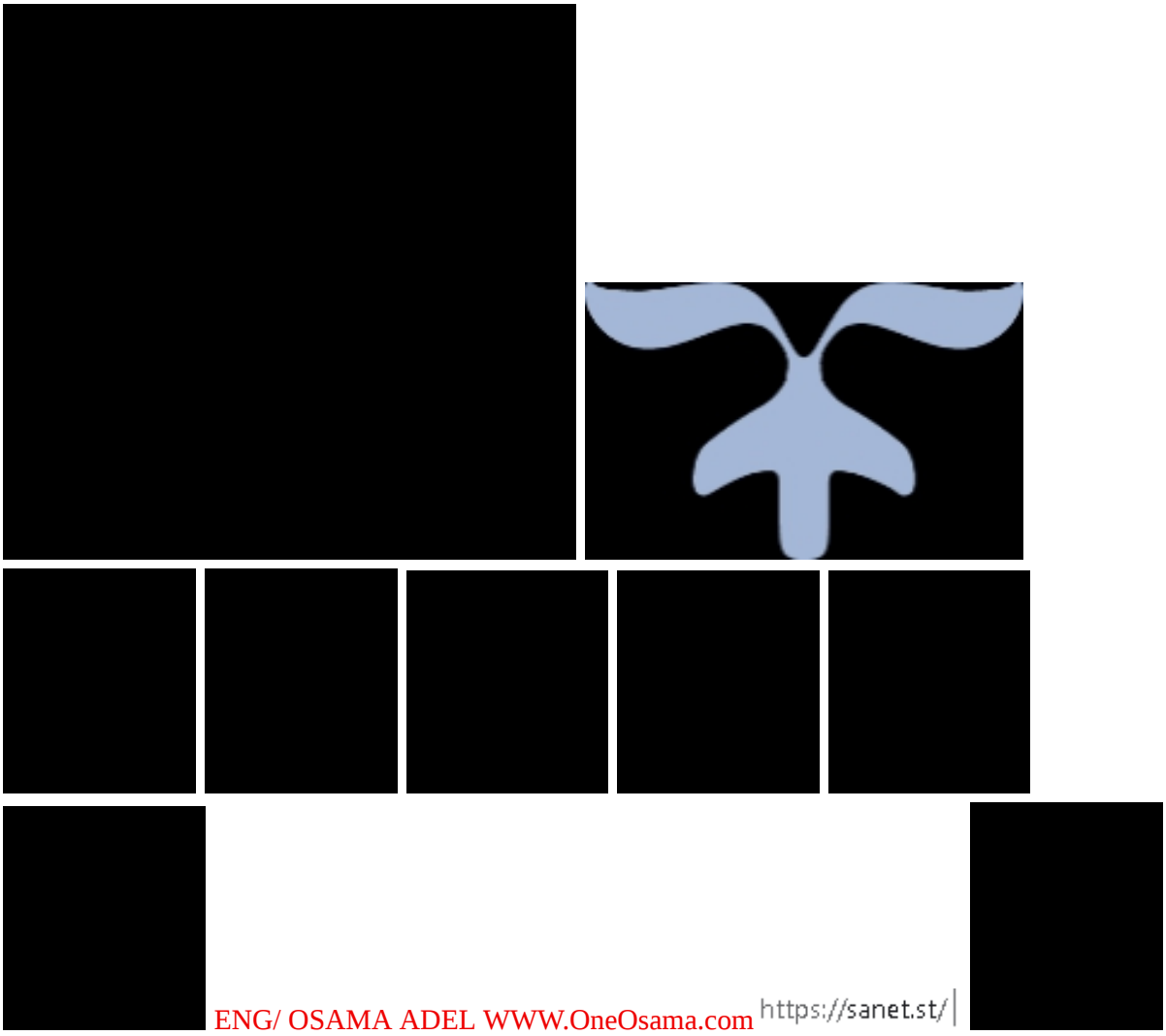
PRO WEB DIRECTOR IN UAE



Learn and Practice Python in 7 Days



From Zero To Hero

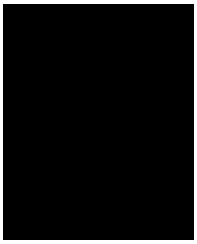


Before starting to learn through this book, you must download

Before starting to learn through this book, you must download

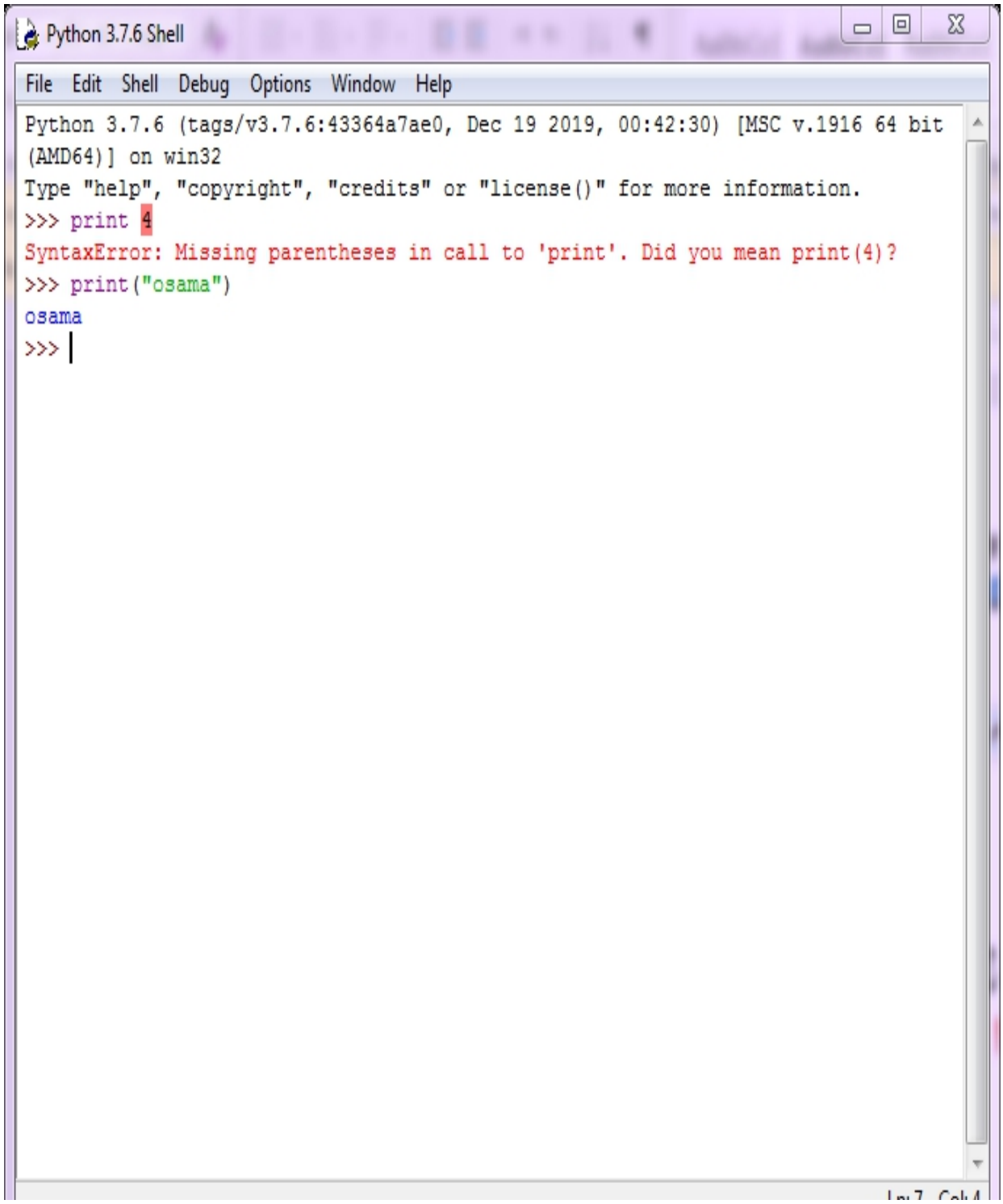
the Python language from the official site and download any editor to write codes or use the official python language editor.

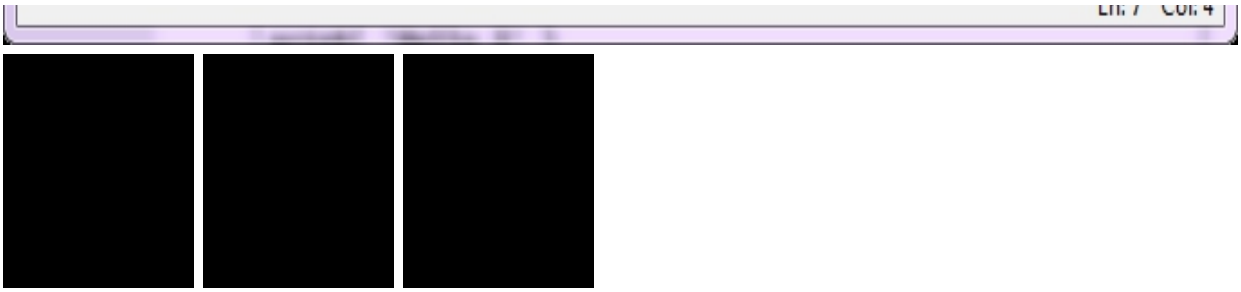
editor to write codes or use the official python language editor.



python language editor.

python language editor.

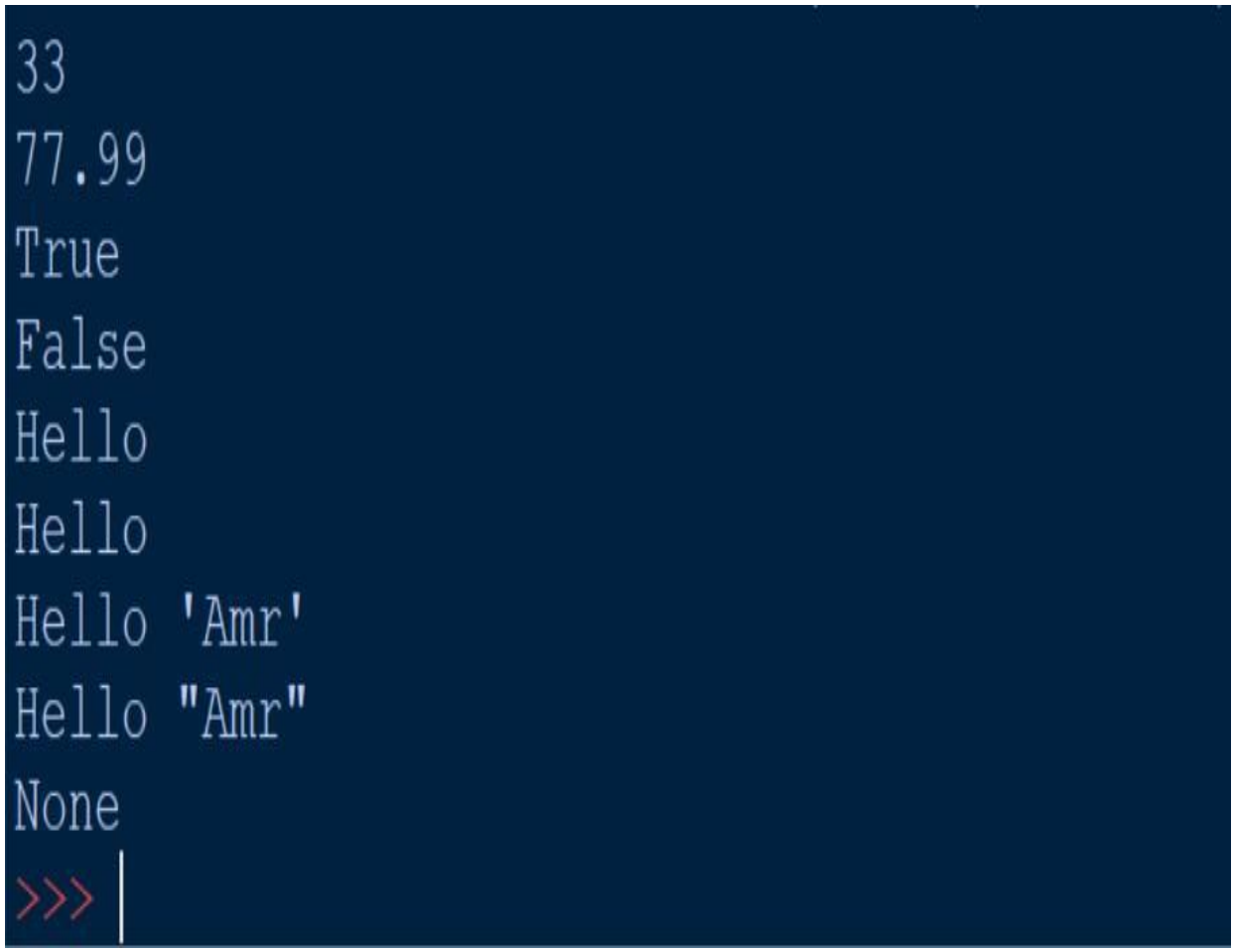




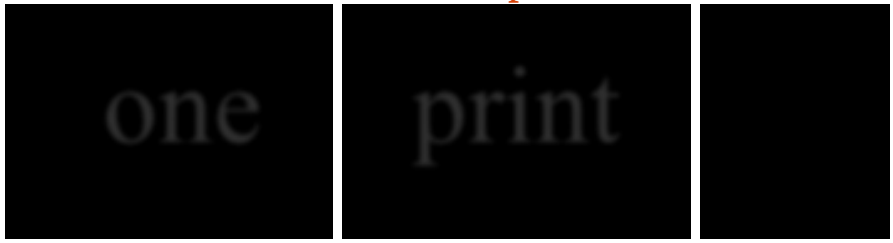
Print Function:

Print different values

print(33) print(77.99)
print(True)
print(False)
print("Hello")
print('Hello')
print("Hello 'Amr'")
print('Hello "Amr"')
print(None)



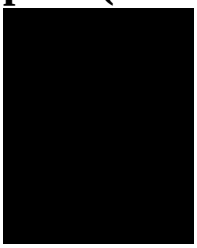
one print:

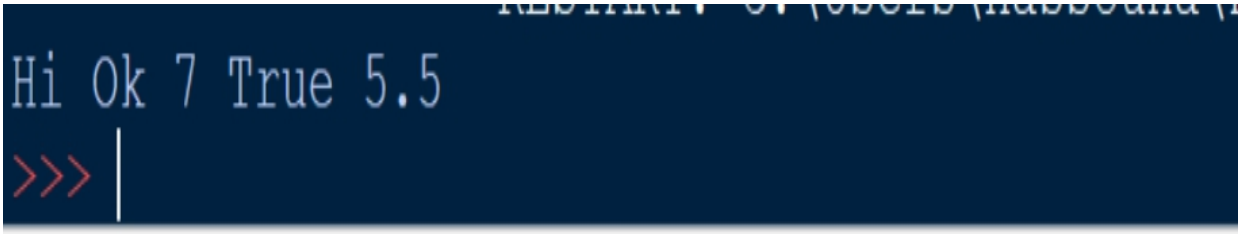


Multiple print in

bulk

print("Hi" , 'Ok' , 7 , True , 5.5)





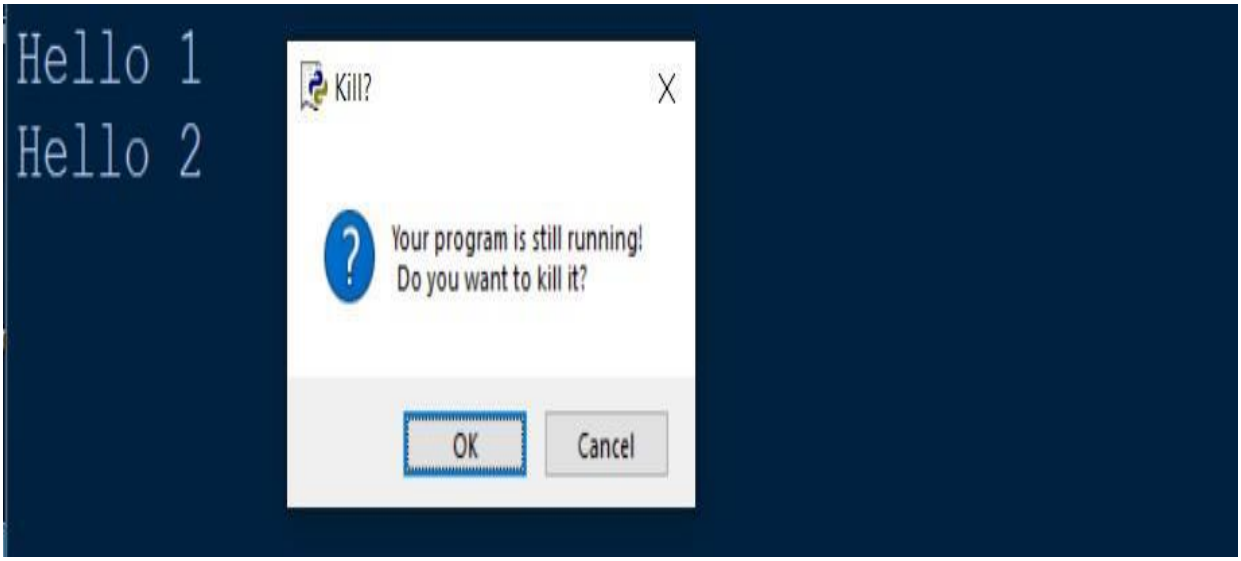
Exit Function

Exit the program and not complete the post-exit

That was the following


```
print( 'Hello 1' )
print( 'Hello 2' )
exit()
print( 'Hello 3' )
```

Implementation



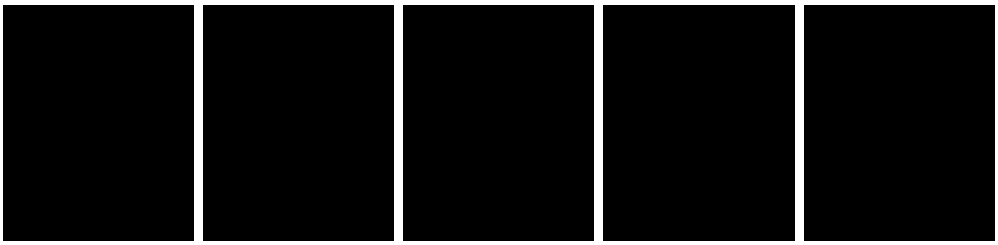
Executing more than one line; For insulation

Executing more than one code using

That was the following

print('Ok 1'); print('Ok 2'); print('Ok 3')

Implementation



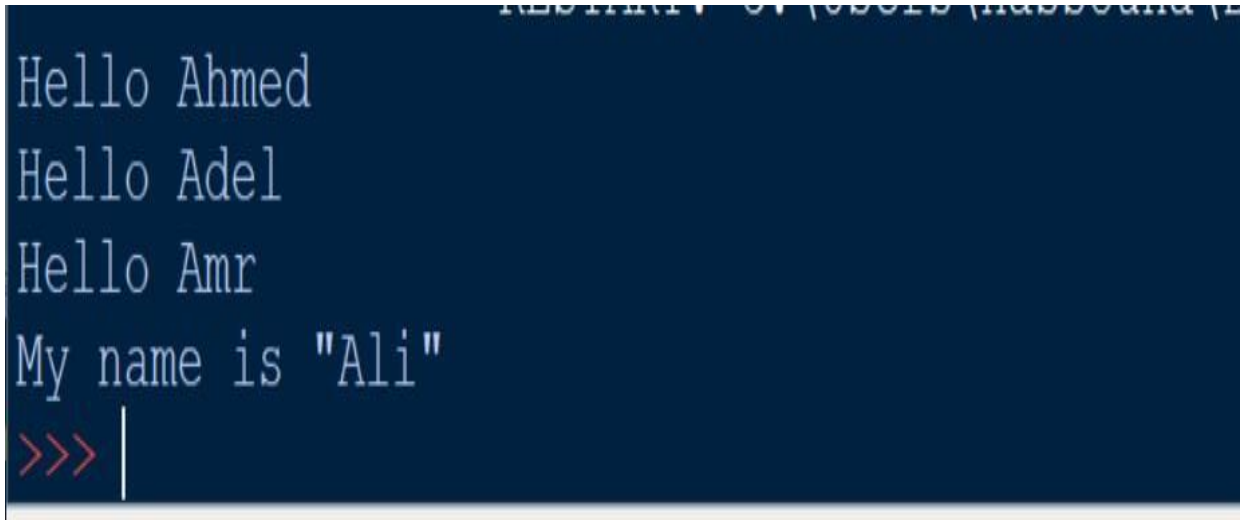
Combine texts with the + symbol

Between texts Merge and paste text with +

That was the following


```
print( "Hello" + " " + "Ahmed" )
print( "Hello" + ' ' + 'Adel' )
print( 'Hello' + ' ' + 'Amr' )
print( 'My' + " " + 'name is ' + '"' + "Ali" + '"' )
```

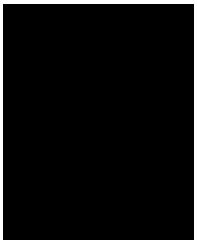
Implementation



Mathematical transactions

Use addition, subtraction, multiplication, and division

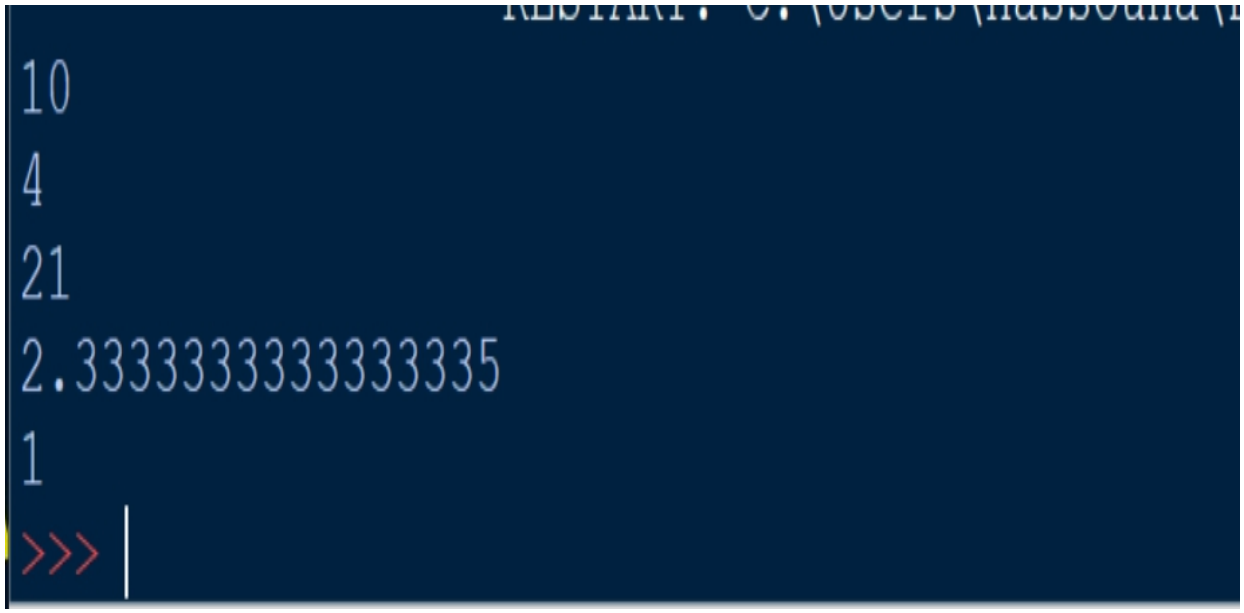
That was the following and the rest of the division Code

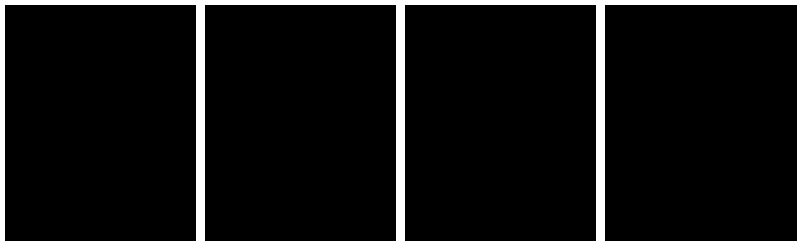


print(7+3)
print(7-3)
print(7*3)
print(7/3)

print(7%3)

Implementation





// fraction can be removed with division using // can

Here it is

Calculating mathematical exponent using **

Code


```
print( 7//3 )
print( 5**3 )
```

Implementation

[redacted] Variables Define two variables and group them into a third variable

[redacted]

[redacted] That was the following

[redacted]

[redacted]

num1 = 7

[redacted]

num2 = 3

result = num1 + num2

print(result)

Implementation

Distance Hello Variable and then combine it with the word

That was the following

```
name = 'Ahmed'
say_hello = 'Hello ' + name
print(say_hello)
```

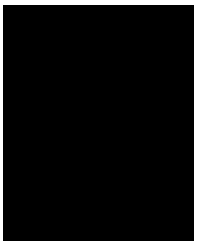

Implementation


Respectively, then give them the values Define variables on

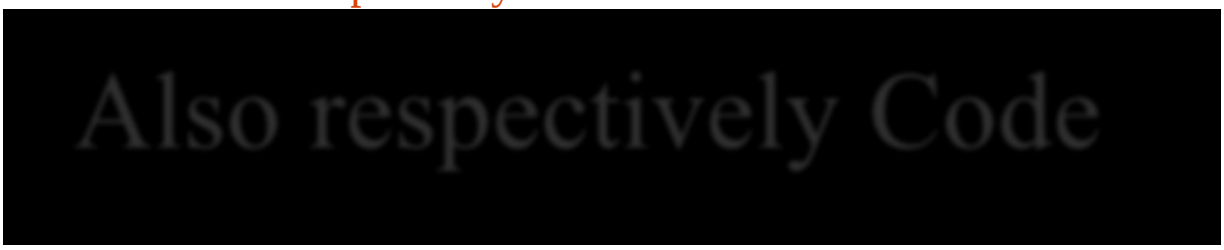
That was the following

name1, name2, name3 = 'Ahmed', 'Adel', 'Amr'
print(name1 + ' ' + name2 + ' ' + name3)

Implementation



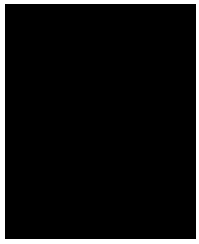
Also respectively Code



num, name, salary, is_active = 1, 'Ali', 4500.66, True print(num)

print(name)
print(salary)
print(is_active)

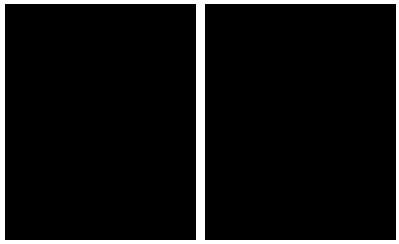
Implementation



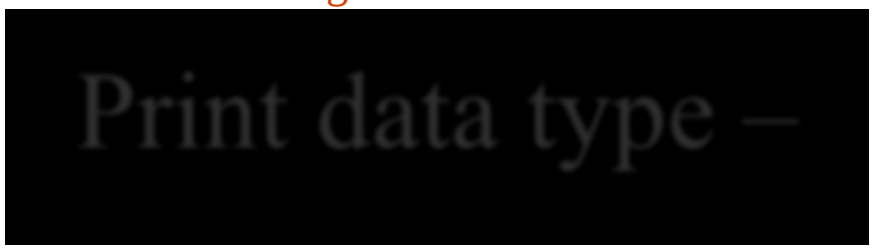
The variable contains only one value Here it is

name = 'Adel'
print(name)
name = 'Ahmed'

name = 'Amr'
print(name)


Implementation


Type Function Print data type – type Show data type for each variable using



That was the following


```
var1 = 733
var2 = 99.55
var3 = True
var4 = 'Hello'
var5 = "Hi"
print( type(var1) )
print( type(var2) )
print( type(var3) )
print( type(var4) )
print( type(var5) )
```

Implementation


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

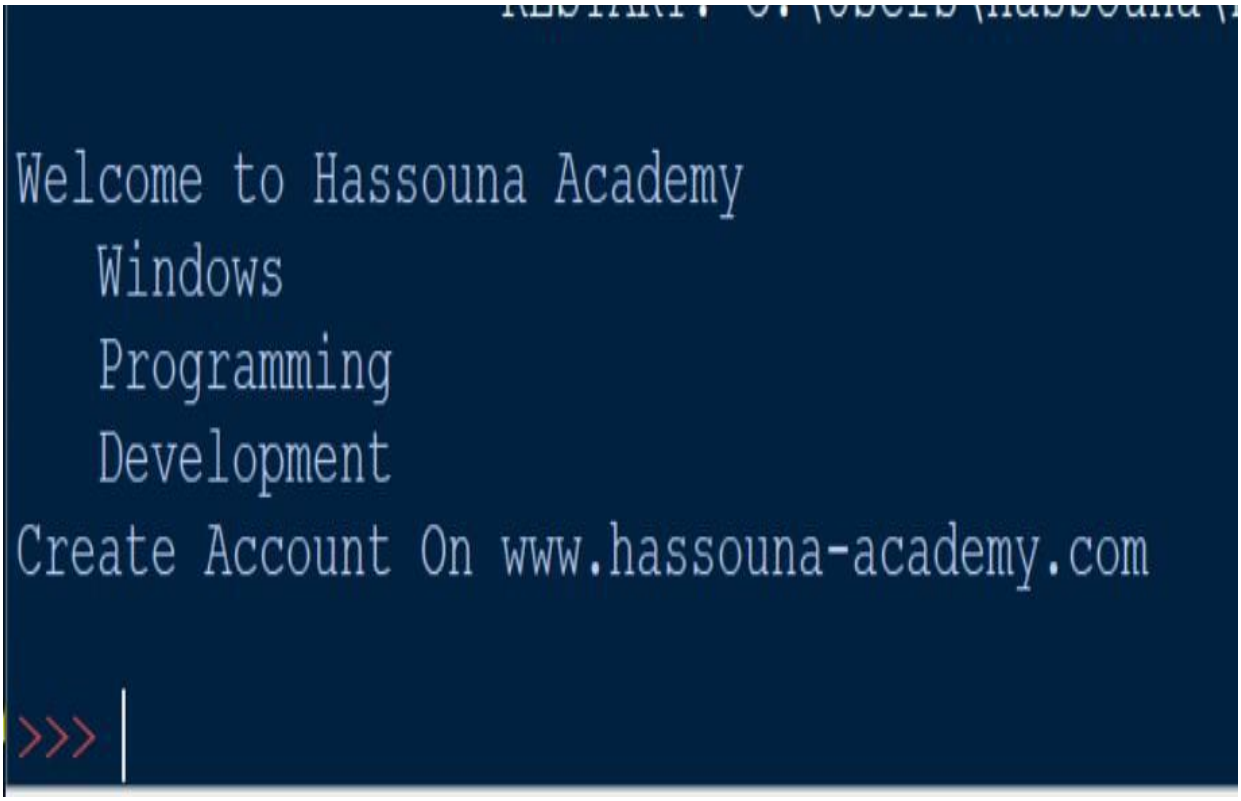
Multiline text variable Define a text variable with text lines

printed as it is

That was the following


```
my_str = """

Welcome to Hassouna Academy
Windows
Programming
Development

Create Account On www.hassouna-academy.com
"""
print(my_str)
```

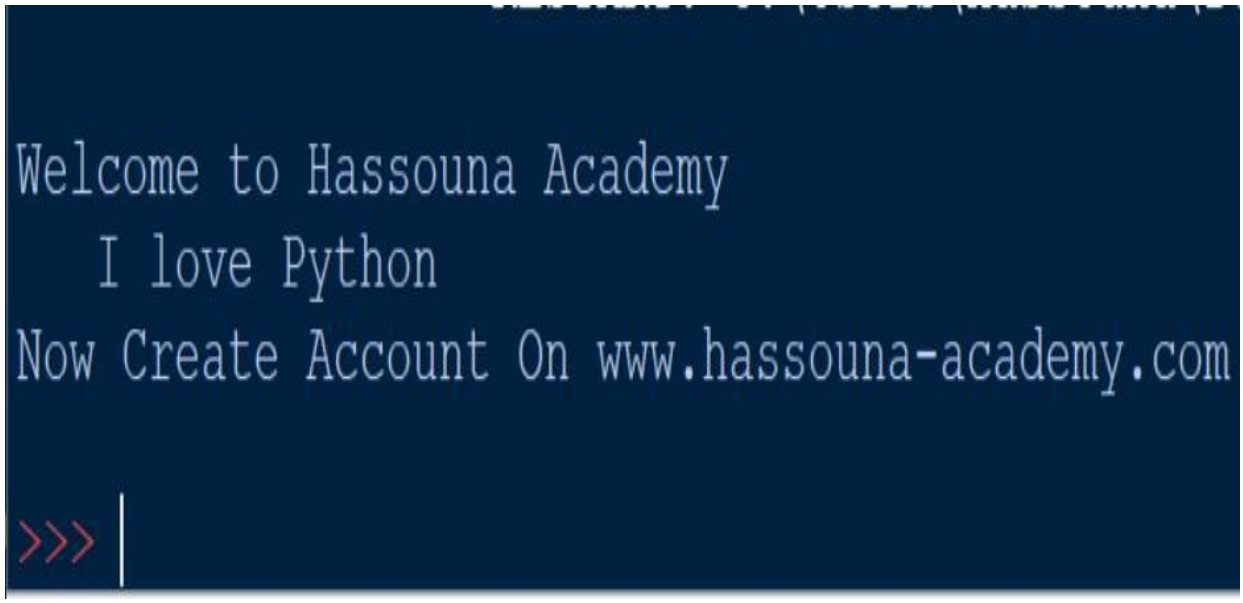
Implementation



Code


```
my_str = '''
Welcome to Hassouna Academy

I love Python
Now Create Account On www.hassouna-academy.com
'''
print(my_str)
```

Implementation



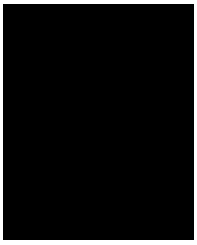
Repetitive text with symbol *

Use the multiplication operator * to repeat text with a certain number

That was the following


```
print( 'A ' * 5 )
name = ' AMR '
print( name * 3 )
```

Implementation



Strings Using strings or what is known as strings

That was the following

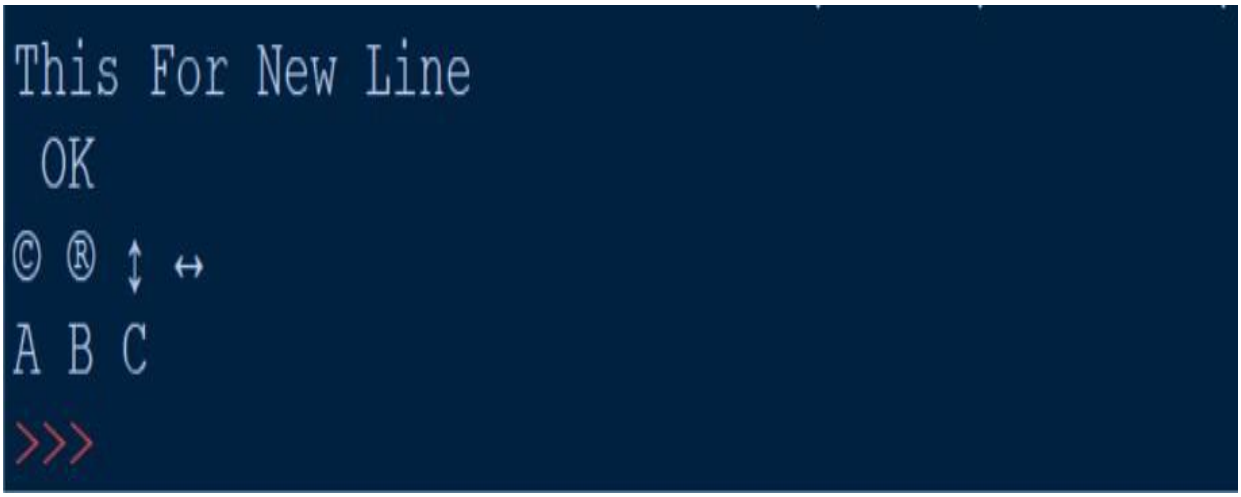
What each one does is explained Escape Sequence

Escape Include them when printing with every code

Code

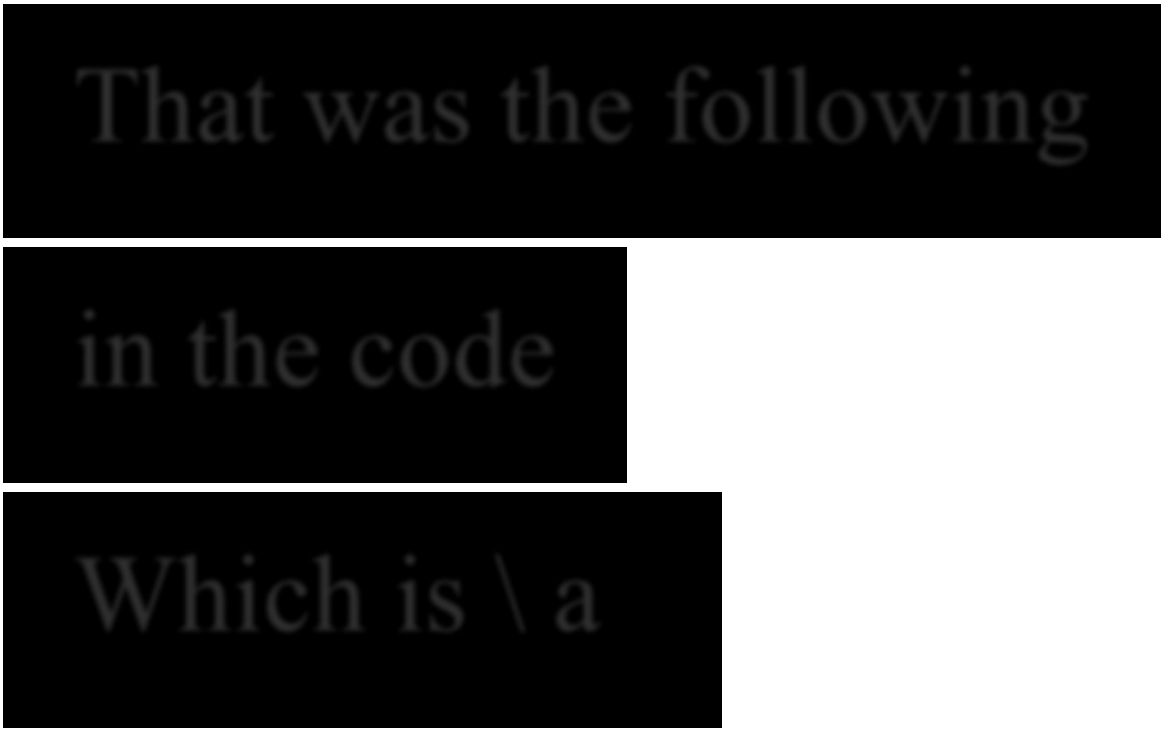

```
str1 = '\n'
str2 = '\N{copyright sign}'
str3 = '\N{registered sign}'
str4 = '\N{up down arrow}'
str5 = '\N{left right arrow}'
str6 = '\x41'
str7 = '\u0042'
str8 = '\U00000043'

print( 'This For New Line' , str1 , 'OK' ) print( str2, str3, str4,str5 )
print( str6, str7, str8 )
```

Implementation

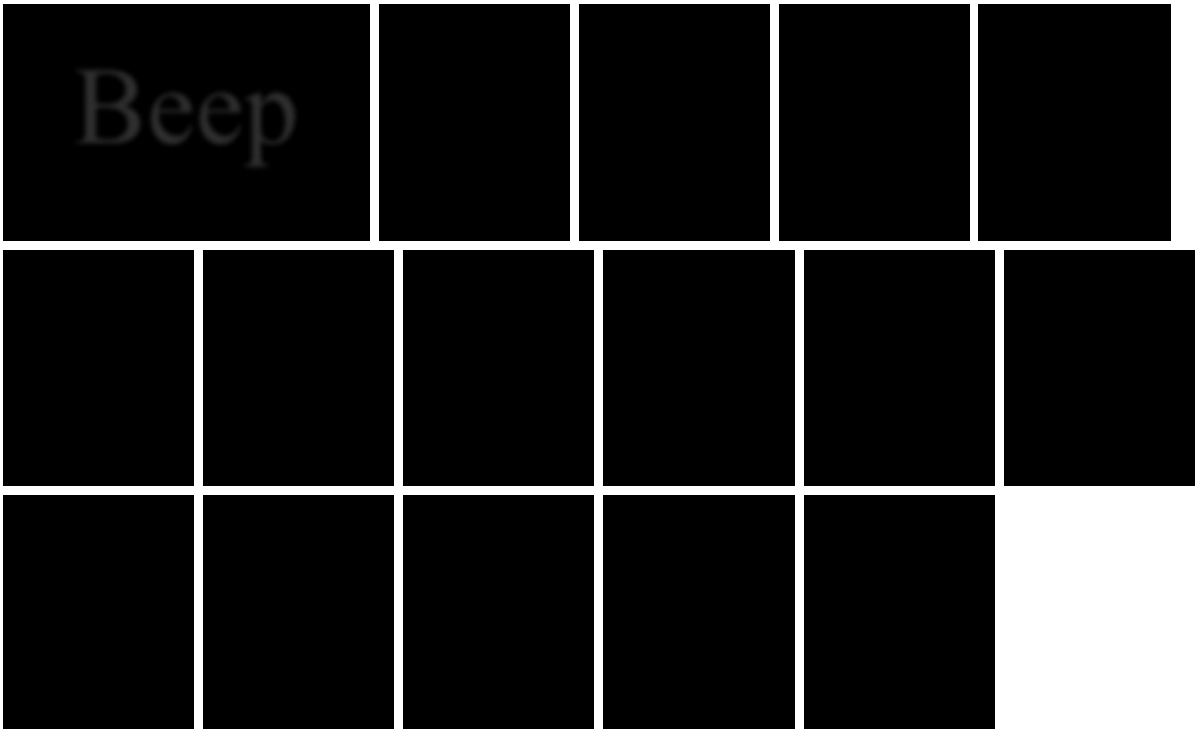


Upon execution to hear the beep DOS Using the

That was the following in the code Which is \ a



Beep

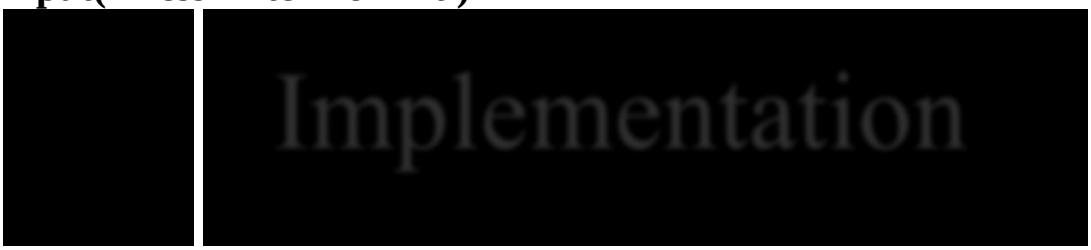


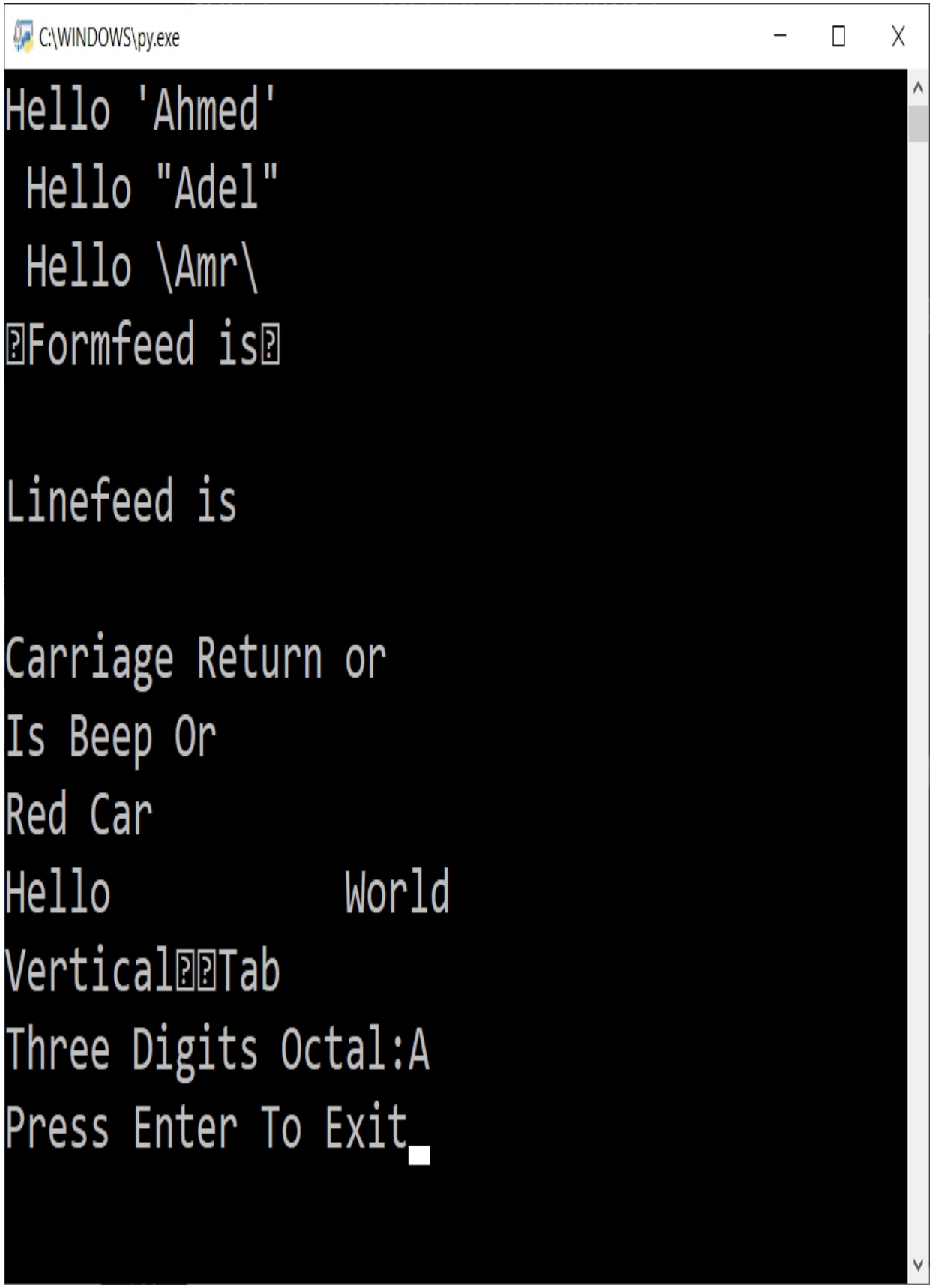
Code


```
str1 = 'Hello \'Ahmed\''
str2 = "Hello \"Adel\""
str3 = 'Hello \\Amr\\'
str4 = '\fFormfeed is\N{FF}'
str5 = '\nLinefeed is\N{LF}'
str6 = 'Welcome to Egypt\rCarriage Return or \N{CR}' str7 = '\aIs Beep Or \N{BEL}'
str8 = 'Rad Cat\br\b\b\b\b\N{BS}\be'
str9 = 'Hello\t\N{TAB}World'
str10 = 'Vertical\v\N{VT}Tab'
str11 = 'Three Digits Octal:\101'
```

print(str1, '\N{LF}', str2, '\n', str3) print(str4)
print(str5)
print(str6)
print(str7)
print(str8)
print(str9)
print(str10)
print(str11)

input('Press Enter To Exit')

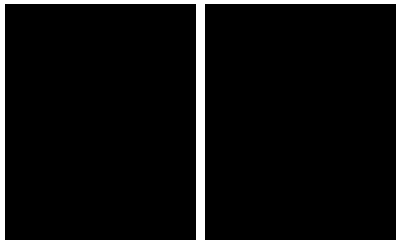



C:\WINDOWS\py.exe
Hello 'Ahmed'
Hello "Adel"
Hello \Amr\
Formfeed is
Linefeed is
Carriage Return or
Is Beep Or
Red Car
Hello World
Vertical Tab
Three Digits Octal:A
Press Enter To Exit

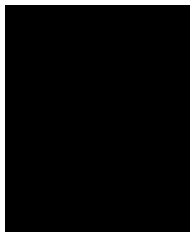
Non-changeable values Tuple The variable It has values and Appeal to them tuple Use a variable

That was the following

p1 = (1,'Ahmed',3900.50)
p2 = (2,'Adel',4600.60)
p3 = (3,'Amr',4500.55)



print(p1); print(p2); print(p3)
print(p1[0], p1[1], p1[2])

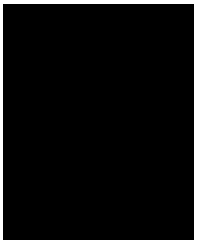


print(type(p1))

print(type(p1[0]), type(p1[1]), type(p1[2]))

Implementation


```
(1, 'Ahmed', 3900.5)
(2, 'Adel', 4600.6)
(3, 'Amr', 4500.55)
1 Ahmed 3900.5
<class 'tuple'>
<class 'int'> <class 'str'> <class 'float'>
>>>
```



Changeable values List the existing variable list with values and appeal list use a variable

That was the following

Index on it with the number

Code


```
numbers = [11,22,33]
names = ['Amr','Ali','Ezz']
```

print(numbers); print(names)
print(numbers[0])
print(numbers[1])
print(numbers[2])
names[0] = 'Akl'
print(names)

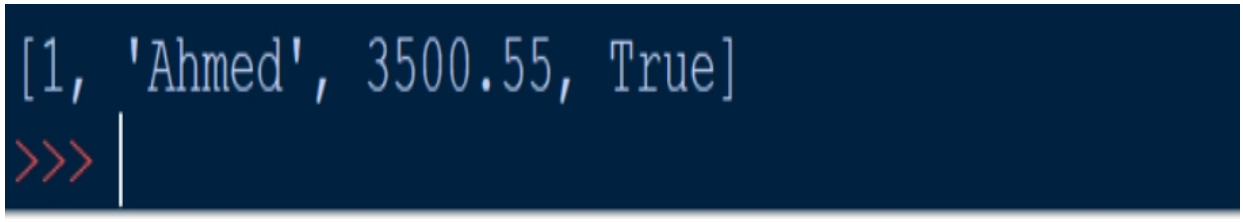
Implementation


```
[11, 22, 33]
['Amr', 'Ali', 'Ezz']
11
22
33
['Akl', 'Ali', 'Ezz']
>>>
```

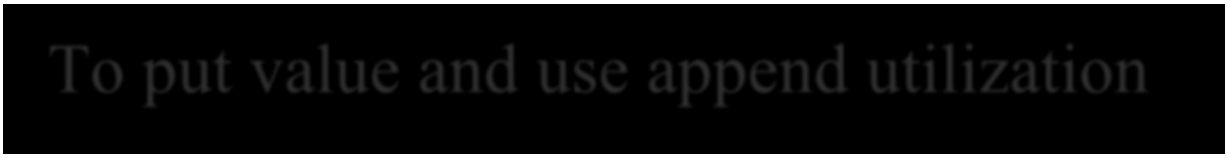
Code

person = [1,'Ahmed',3500.55,True]
print(person)

Implementation



To put value and use append utilization



That was the following

To delete a value remove single payment and use values

To put extend copy to put a value in a specific place and use

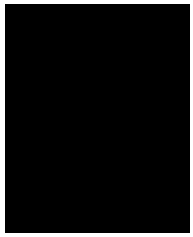
Insert and use to delete the values clear to copy values and use

Code

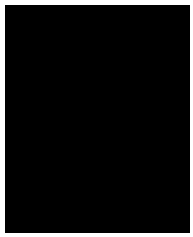

```
names = ['Amr','Ali','Ezz']
print(names)
names.append('Omar')
print(names)

names.extend( ['Adel','Akl'] )
print(names)
names.remove(names[1])
print(names)
names.insert(1,'Ali')
```

print(names)

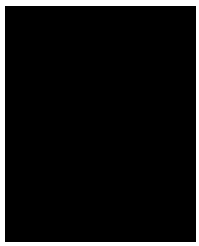
names2 = names.copy()



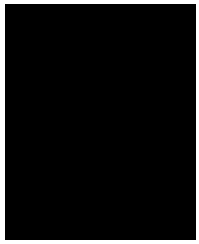
names.clear()



print(names)
print(names2)



```
['Amr', 'Ali', 'Ezz']
['Amr', 'Ali', 'Ezz', 'Omar']
['Amr', 'Ali', 'Ezz', 'Omar', 'Adel', 'Akl']
['Amr', 'Ezz', 'Omar', 'Adel', 'Akl']
['Amr', 'Ali', 'Ezz', 'Omar', 'Adel', 'Akl']
[]
['Amr', 'Ali', 'Ezz', 'Omar', 'Adel', 'Akl']
>>>
```

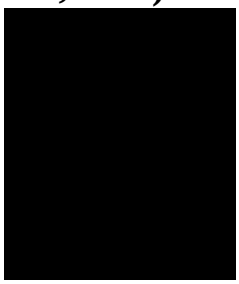


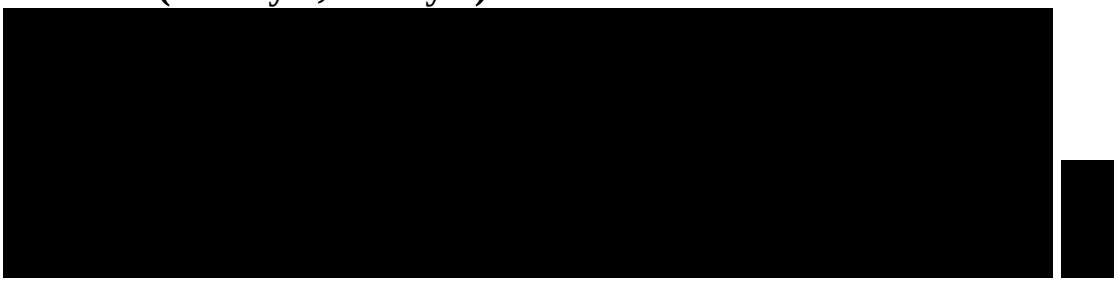
Nested row variable branches off from it tuples do a variable loaded with a dash inside

family1 = ('Ahmed','Adel','Amr')

family2 = ('Sarah','Hajer','Rehab')
family3 = ('Tawfeek','Ezzat','Foaad')

family4 = ('Hasan','Shokry','Ali','Akl')

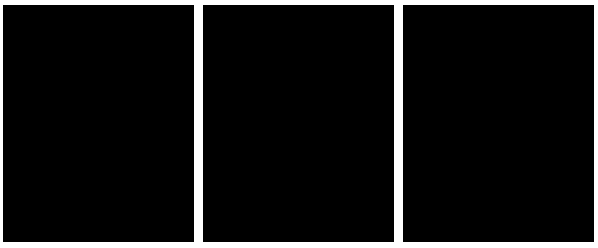
home1 = (family1 , family2) 



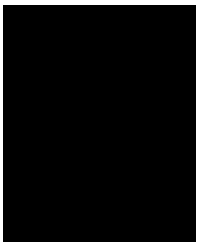
home2 = (family3 , family4)

print(home1[0][1]) print(home1[1][0])

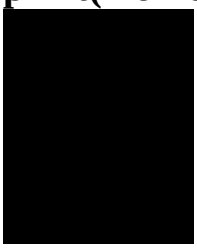
print(home2[0][2])



print(home2[1][3])



print(home1[0])

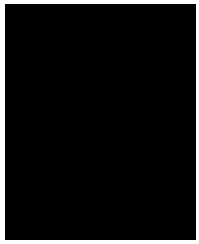


print(home2[1])

print(home1)

print(home2)

```
Adel
Sarah
Foaad
Akl
('Ahmed', 'Adel', 'Amr')
('Hasan', 'Shokry', 'Ali', 'Akl')
(('Ahmed', 'Adel', 'Amr'), ('Sarah', 'Hajer', 'Rehab'))
(('Tawfeek', 'Ezzat', 'Foaad'), ('Hasan', 'Shokry', 'Ali', 'Akl'))
>>>
```



Variable nested lists Branch off from it lists do a variable loaded with a dash inside

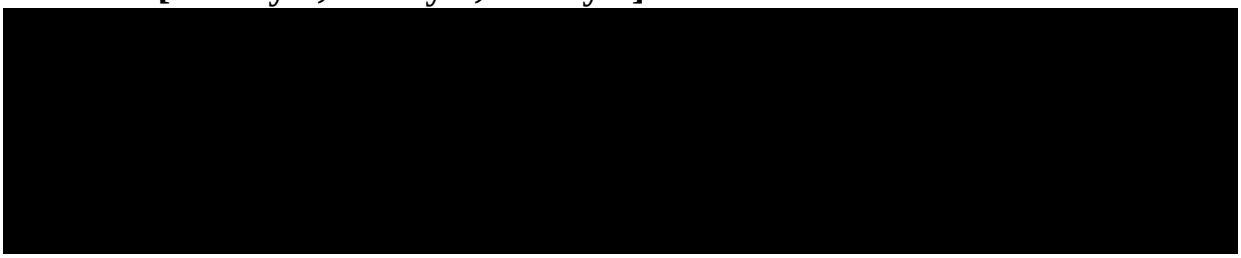
That was the following

family1 = ['Ahmed','Adel','Amr']
family2 = ['Ehab','Mahmoud','Ezz']

family3 = ['Sarah','Hajer','Rehab']

family4 = ['Tawfeek','Ezzat','Foaad'] family5 = ['Abdelrahman','Abdelkareem'] family6 = ['Hasan','Shokry','Ali','Akl']

home1 = [family1 , family2 , family3]



home2 = [family4 , family5 , family6]

print(home1[0][1])
print(home1[1][0])

print(home2[0][2])

print(home2[2][3])
print(home1[0])
print(home2[1])
del(home1[1])

del(home2[1])
print(home1)

print(home2) 18

```
Adel
Ehab
Foaad
Akl
['Ahmed', 'Adel', 'Amr']
['Abdelrahman', 'Abdelkareem']
[['Ahmed', 'Adel', 'Amr'], ['Sarah', 'Hajer', 'Rehab']]
[['Tawfeek', 'Ezzat', 'Foaad'], ['Hasan', 'Shokry', 'Ali', 'Akl']]
>>>
```

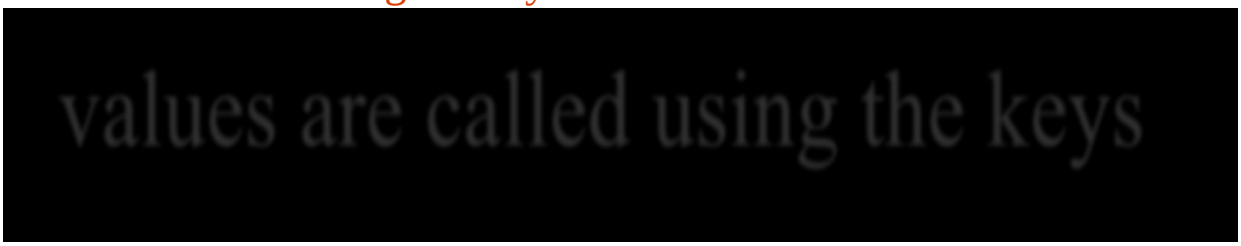
The dictionary variable use variables of special type which is

The dictionary variable use variables of special type which is

That was the following

Value and value key as each value holds a key dictionary the

values are called using the keys



Code


```
person1 = {'name':'Amr', 'salary':5000, 'active':True} person2 = {'name':'Ali', 'salary':4000, 'active':True} person3 = {'name':'Ezz', 'salary':3000, 'active':True}

print( person1['name'], person1['salary'], person1['active'] ) print( person2['name'], person2['salary'], person2['active'] ) print( person3['name'], person3['salary'], person3['active'] )
```

Implementation

Show keys and values that was the following

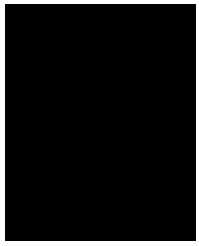
person = {'name':'Amr', 'salary':5000, 'active':True} print(person.keys())
print(person.values())

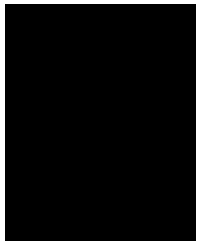
Implementation

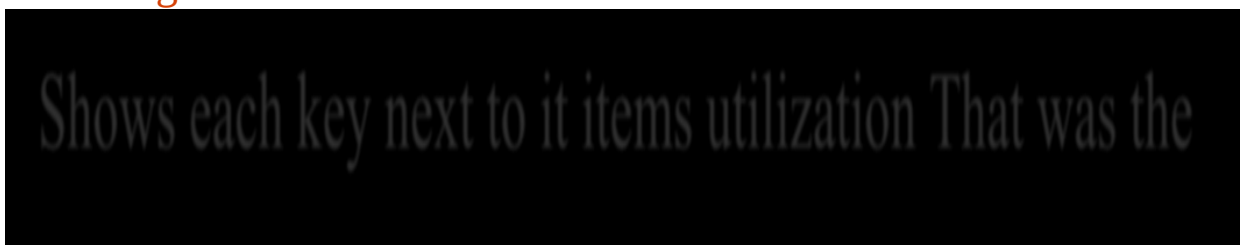

```
dict_keys(['name', 'salary', 'active'])
dict_values(['Amr', 5000, True])
>>>
```

List Keys and values can be made up of Here it is

person = {'name':'Amr', 'salary':5000, 'active':True} print(list(person.keys()))
print(list(person.values()))


Implementation

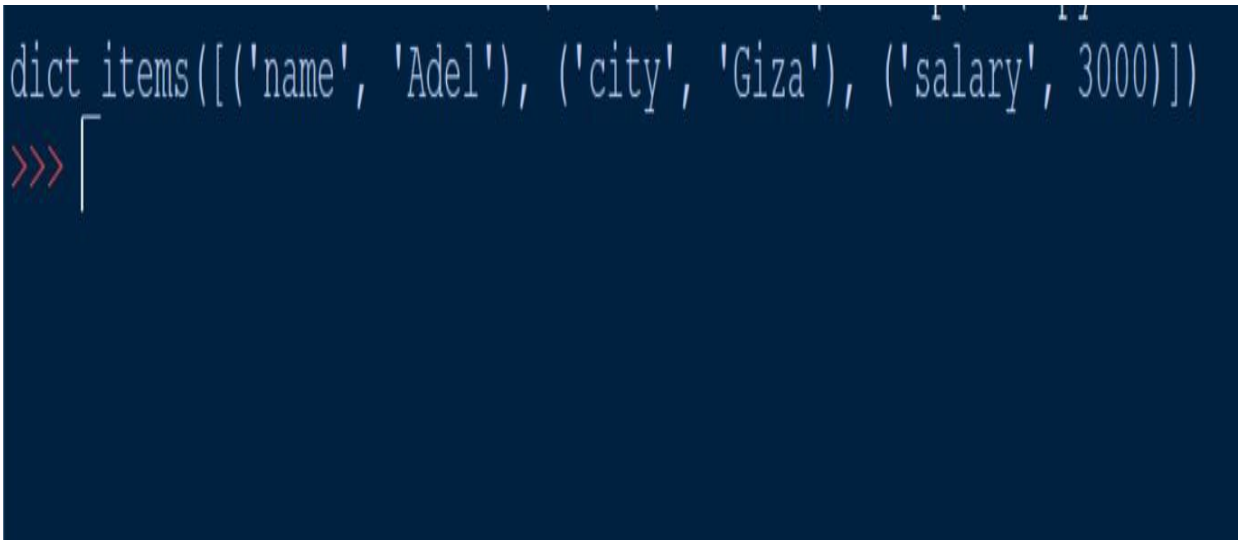

Shows each key next to it items utilization That was the following



Its value, and can be made a list

person = { 'name':'Adel', 'city':'Giza', 'salary':3000 } print(person.items())

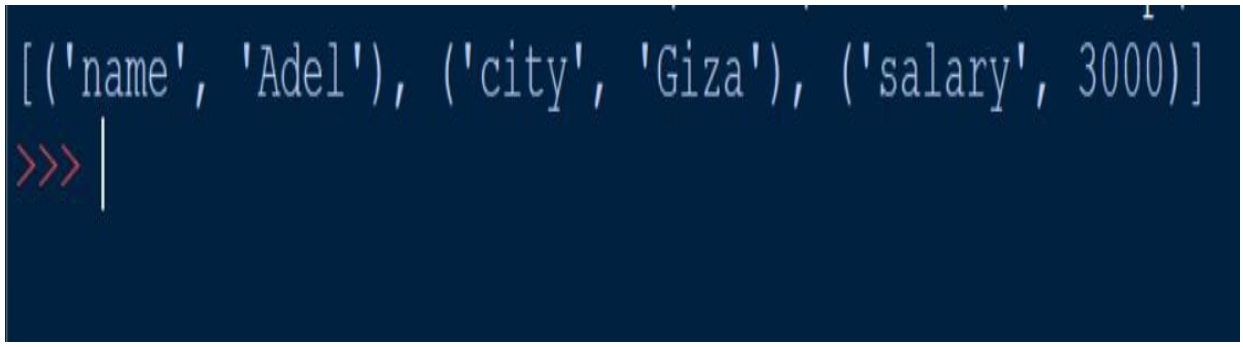
Implementation

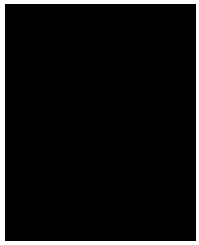


Code

person = { 'name':'Adel', 'city':'Giza', 'salary':3000 } print(list(person.items()))

Implementation




Compound variable using set empty without values set utilization

That was the following

names = set()

print(names)

Implementation

Set Put a list inside that was the following

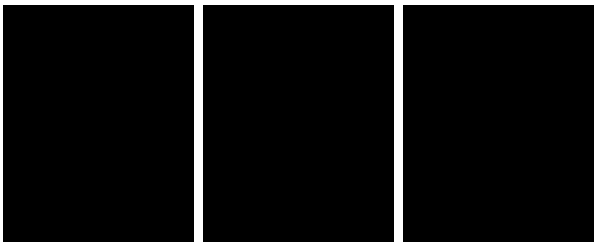
L1 = ['Ahmed','Adel','Omar']
names = set(L1)
print(names)

Implementation

```
{'Omar', 'Ahmed', 'Adel'}
>>>
```

Update create a new list with that was the following


```
L1 = ['Ahmed','Adel','Omar']
L2 = ['Amr','Ali','Ezz']
names = set(L1)
names.update(L2)
print( names )
```


Implementation


```
{'Amr', 'Adel', 'Ezz', 'Omar', 'Ahmed', 'Ali'}
>>>
```

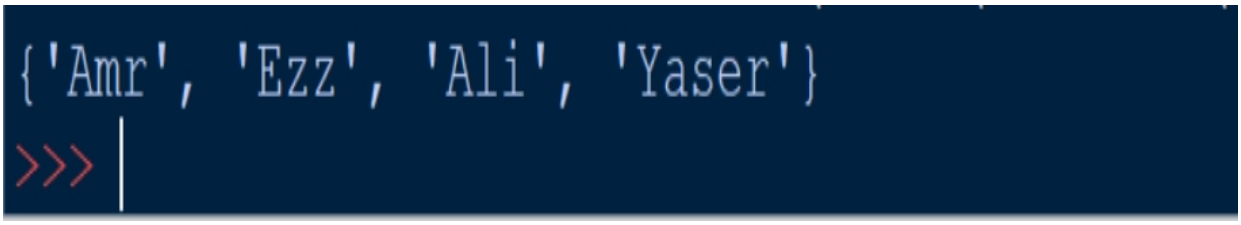
[redacted] To add value add utilization

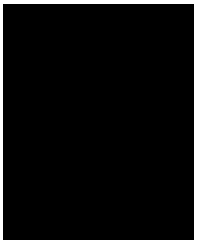
That was the following

[redacted]

L1 = ['Amr','Ali']
names = set(L1)
names.add('Ezz')
names.add('Yaser')
print(names)

Implementation



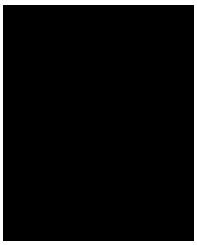


Repeat values, then repetition is not accepted

That was the following


```
names = set()
names.add('Ahmed')
names.add('Ahmed')
names.add('Ahmed')
print( names )
```

Implementation



To delete a value, if only the value remove utilization

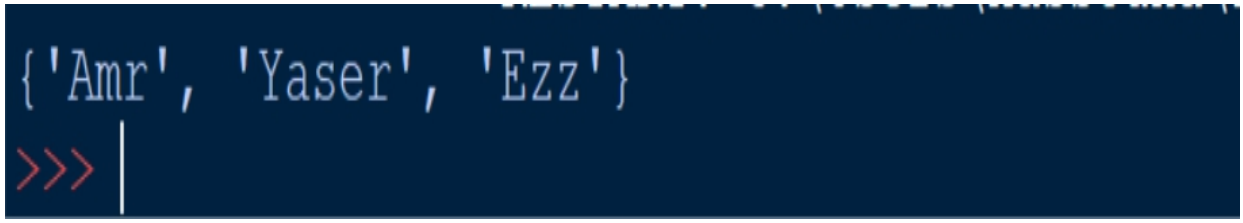
That was the following

To be deleted is not present, the program will generate an error


```
L1 = ['Amr','Ali']
names = set(L1)
names.add('Ezz')
names.add('Yaser')
names.remove('Ali')
print( names )
```

[redacted] Implementation

[redacted]



[redacted]

To delete a value, if only the value discard Utilization

[redacted]

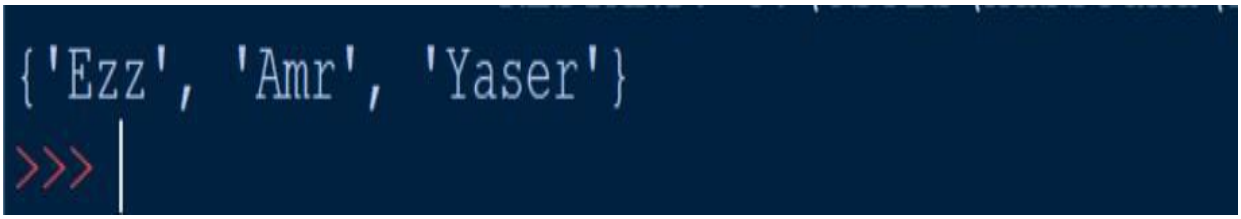
[redacted] [redacted]

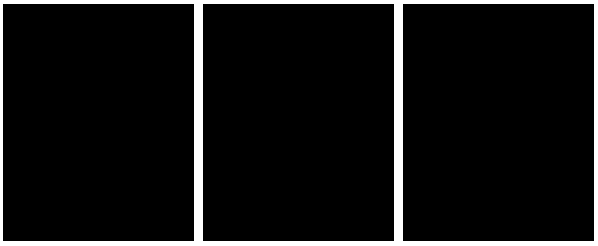
That was the following

The target to be deleted is not found, the program will not generate an error Code


```
L1 = ['Amr','Ali']
names = set(L1)
names.add('Ezz')
names.add('Yaser')
names.discard('Ali')
print( names )
```

Implementation





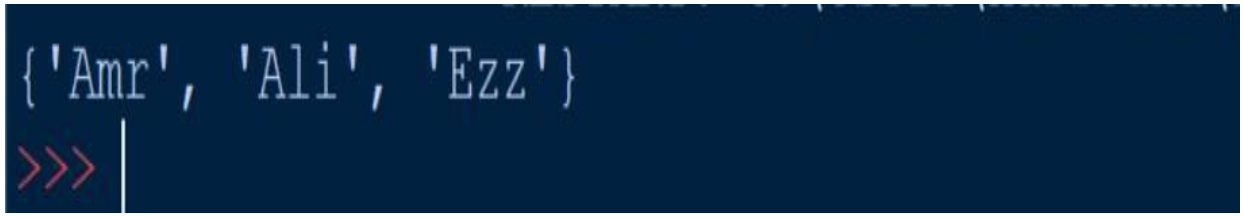
Below, pop was used to delete the last value, without regard

The arrangement, as the arrangement is not fixed

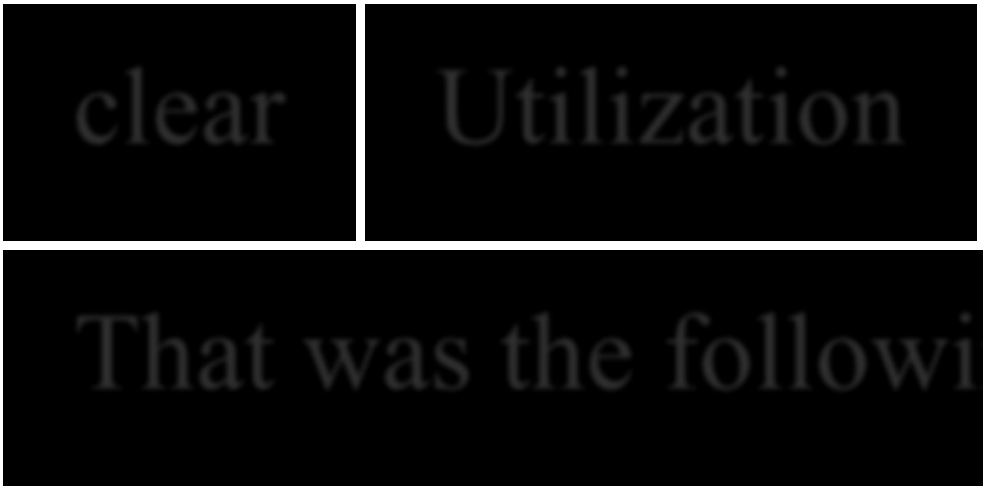
Code

```
L1 = ['Amr','Ali']
names = set(L1)
names.add('Ezz')
names.add('Yaser')
names.pop() #Remove the last item without sort
print( names )
```


Implementation




To delete all values clear Utilization That was the following



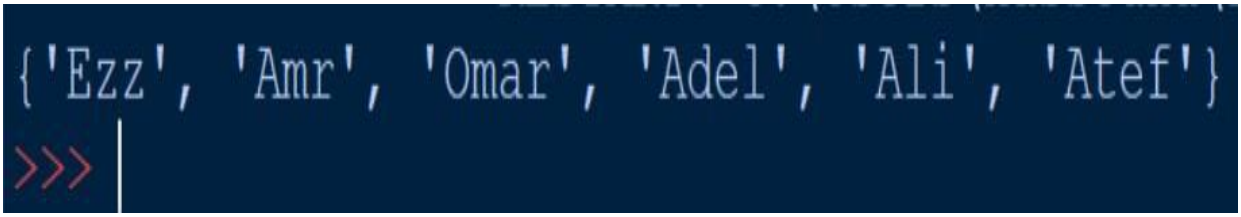

```
L1 = ['Amr','Ali']
names = set(L1)
names.add('Ezz')
names.add('Yaser')
names.clear() #Remove all items
print( names )
```

Implementation

To make a union Utilization That was the following


```
names1 = set(['Adel','Omar','Atef'])
names2 = set(['Amr','Ali','Ezz'])
all_names = names1.union(names2)
print( all_names )
```

Implementation



To make a union, and note that repetition use union

That was the following

Code


```
names1 = set(['Adel','Omar','Atef','Amr'])
```

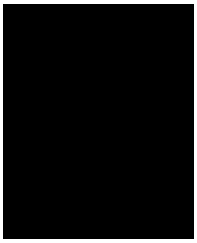
```
names2 = set(['Amr','Ali','Omar','Ezz','Adel'])
```

```
all_names = names1.union(names2)
```

```
print( all_names )
```

Implementation


```
{'Ezz', 'Adel', 'Amr', 'Atef', 'Ali', 'Omar'}
>>>
```



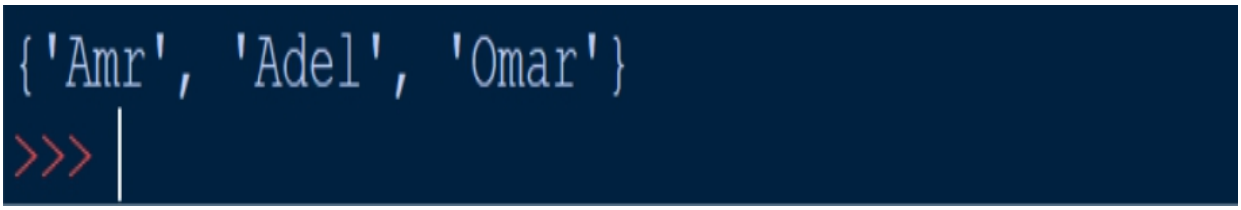
To assemble the subscriber intersection utilization

That was the following

Among them is the code


```
names1 = set(['Adel','Omar','Atef','Amr']) names2 =
set(['Amr','Ali','Omar','Ezz','Adel'])
all_names = names1.intersection(names2)
print( all_names )
```

Implementation



For a different assembly difference utilization

That was the following

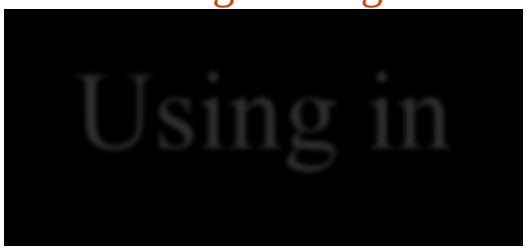

```
names1 = set(['Adel','Omar','Atef','Amr'])
names2 = set(['Amr','Ali','Omar','Ezz','Adel'])
```

```
all_names1 = names1.difference(names2)
all_names2 = names2.difference(names1)
print( all_names1 )
```

```
print( all_names2 )
```

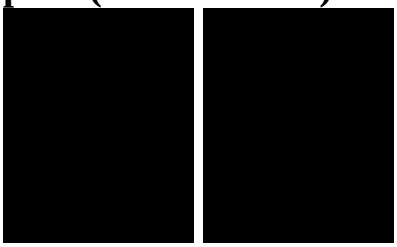
Implementation

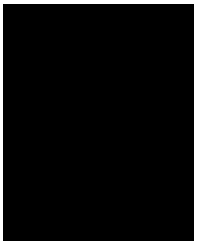

Using in things search in find the character within the text



That was the following

name = 'Ahmed Hassouna'
print('H' in name)


Implementation



Search for the item inside the list that was the following

names = ['Amr','Ali','Ezz']
print('Amr' in names)

Implementation

Search for the item within the group

That was the following

names = {'Amr','Ali','Ezz'}
print('Amr' in names)

Implementation

Search for the item inside the class

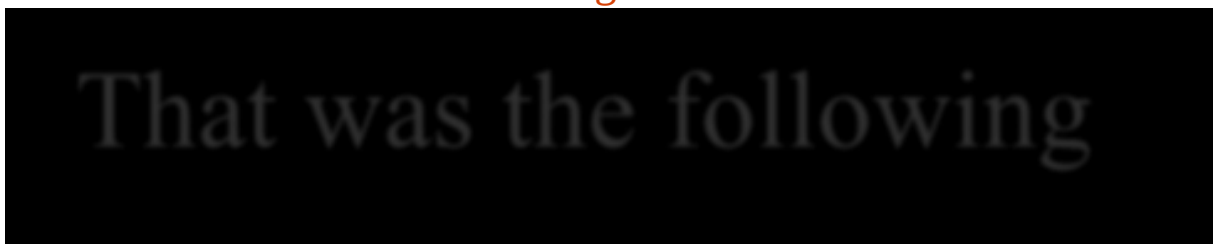
That was the following

names = ('Amr','Ali','Ezz')
print('Amr' in names)

Implementation

Set Search for the item inside the

That was the following

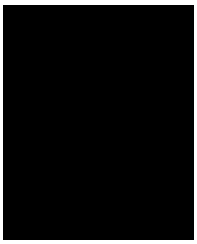



```
names = set(['Amr','Ali','Ezz'])
print( 'Amr' in names )
```

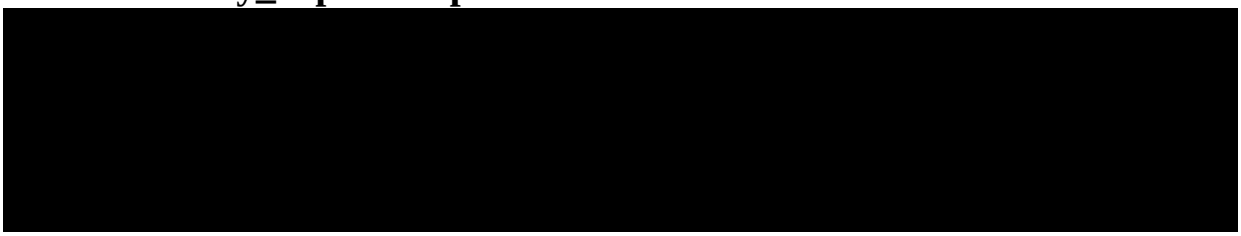
Implementation

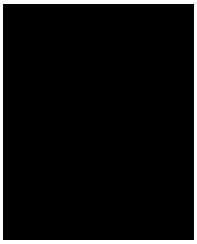
Search for a number in the text within the text

That was the following



my_dept = 'Department 3'





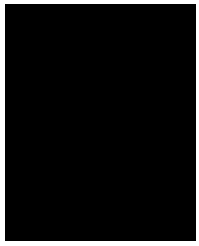
print('3' in my_dept)

Implementation


Del Function things delete a variable, in the same way anything can be deleted

That was the following


```
name = 'Amr'
print( name )
del name
```

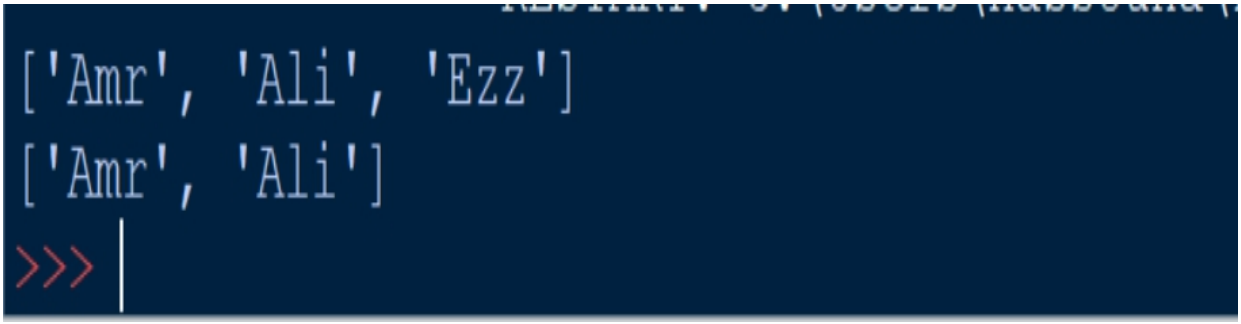


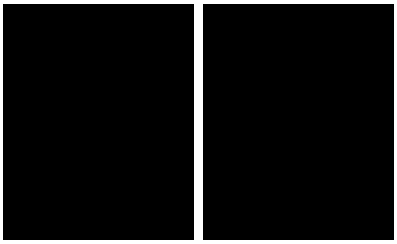
Implementation

Code


```
names = ['Amr','Ali','Ezz']
print(names)
del(names[2])
print(names)
```

Implementation





Len function number of things the process of counting the letters

of the text and knowing the number of elements

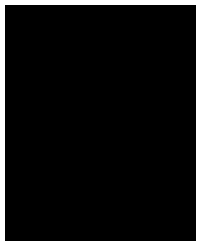
name1 = 'Ahmed'
name2 = 'Adel'
name3 = 'Amr'

names1 = ('Ahmed','Adel','Amr')
names2 = ['Sarah','Hajer','Rehab','Heba']

length_name1 = len(name1) length_name2 = len(name2) length_name3 = len(name3)

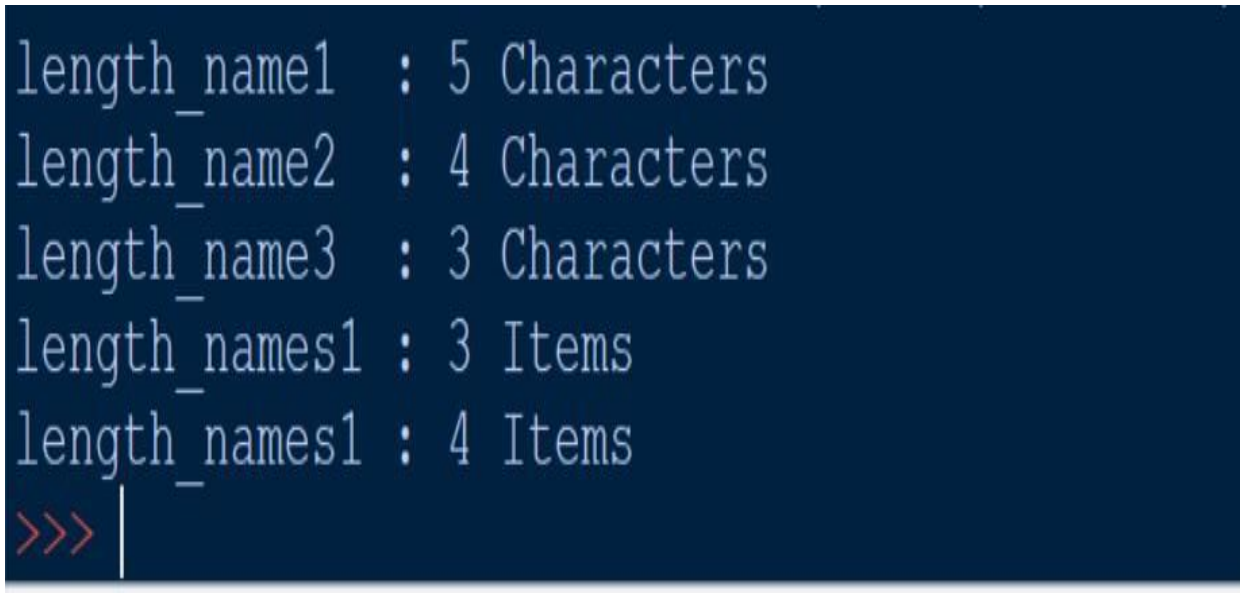
length_names1 = len(names1) length_names2 = len(names2)

print('length_name1 :', length_name1, 'Characters') print('length_name2 :', length_name2, 'Characters') print('length_name3 :', length_name3, 'Characters')

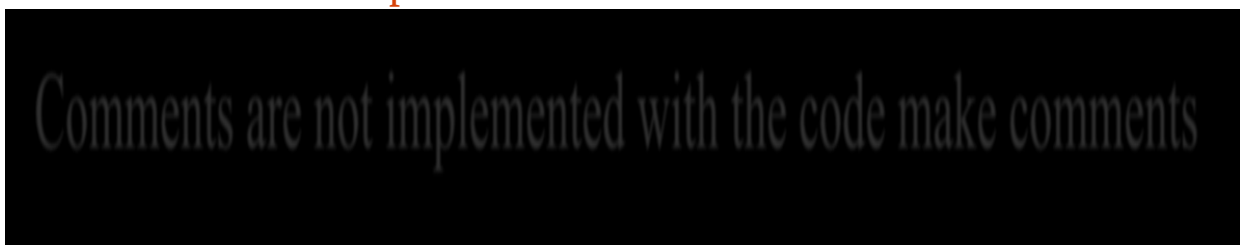


print('length_names1 :', length_names1, 'Items') print('length_names1 :', length_names2, 'Items')

Implementation



Comments are not implemented with the code make comments

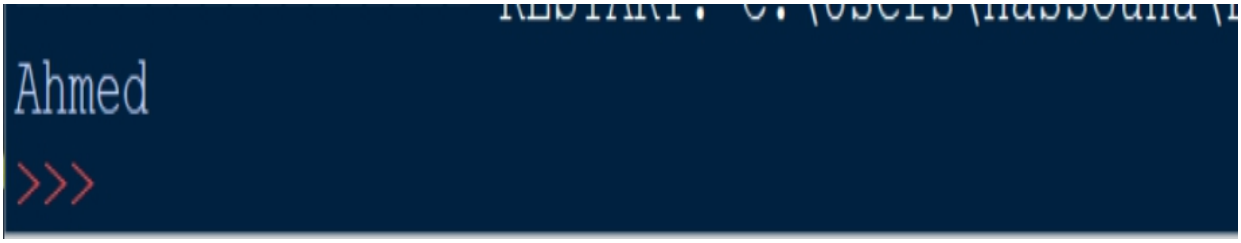


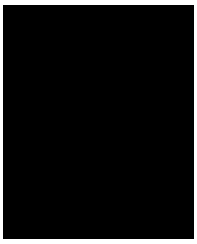
as notes and do not implement with the code

That was the following

#This Words Not Run, But Comment
name = 'Ahmed' #This is my name
print(name) #Print name in here
#The Hash Symbol For One Line Comment
#Can be multiline using # in each first line '''
And Can be multiline using triple single quote In First Paragraph
And In Last Paragraph
'''
"""
And Can be multiline using triple double quote In First Paragraph
And In Last Paragraph
"""

Implementation




30

Allocation factors the addition

Use = operator and allocation coefficients

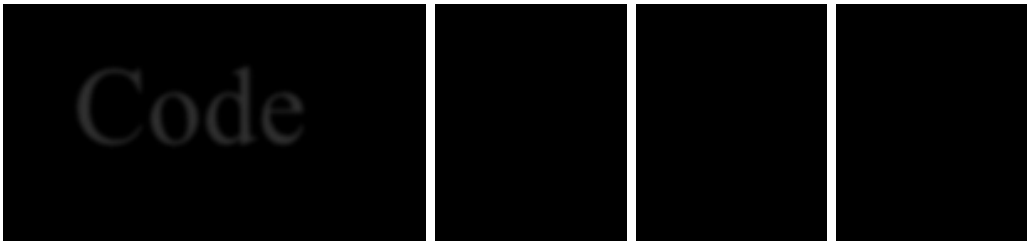
That was the following

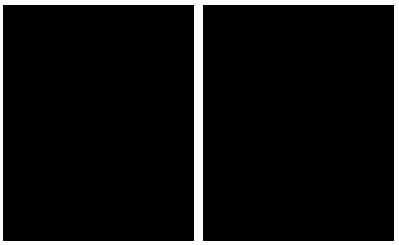
Like the parameter + = i.e. make the variable the same and

increase it, or the parameter / =

That is, make the variable the same and divide it, and so on

Code

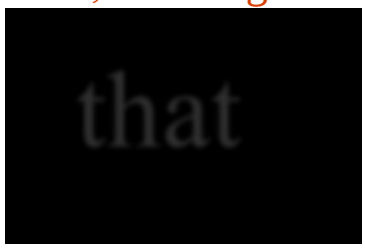


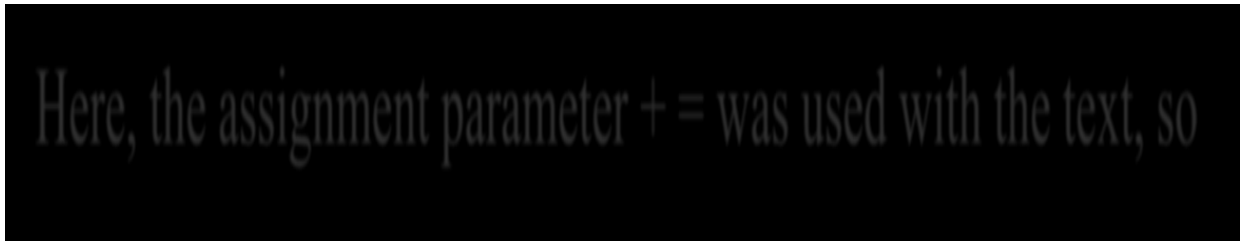


Implementation

```
5
11
7
14
196
65.33333333333333
21.0
9.0
>>>
```

Here, the assignment parameter + = was used with the text, so that





Preserves the existing text and then increases it

```
num = 5; print(num)
num += 6; print(num)
num -= 4; print(num)
num *= 2; print(num)
num **= 2; print(num)
num /= 3; print(num)
num //= 3; print(num)
num %= 12; print(num)
```

```
my_str = ""
my_str += "Hello"
my_str += " "
my_str += "Ahmed"
print(my_str)
```

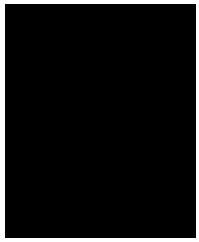
Implementation

[redacted] For several variables allocate one value

That was the following

num1 = num2 = num3 = 7
print(num1)
print(num2)
print(num3)

[redacted] Implementation


Str function convert from number to text -until str convert the

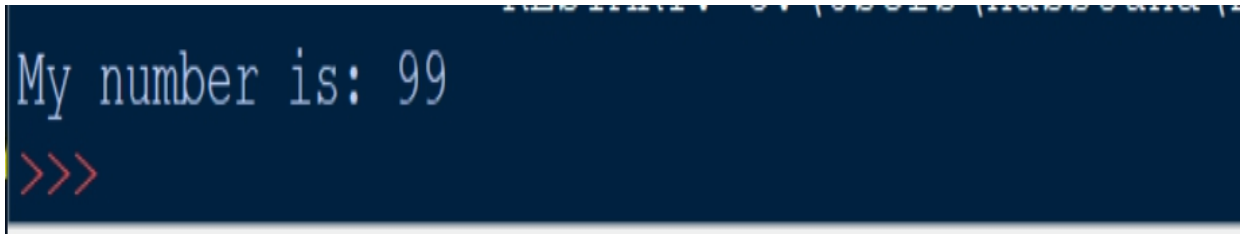
number to text using the function

That was the following

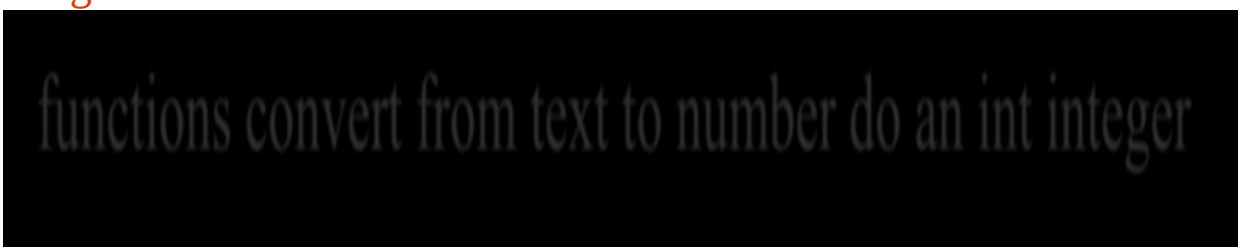
We can combine it with another text

num = 99
print('My number is: ' + str(num))

Implementation



int, float functions convert from text to number do an int integer



conversion and float fraction number

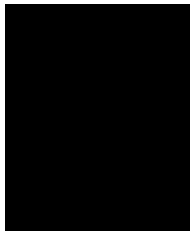

```
num_1 = '77'
num_2 = '77.33'
num1 = int(num_1)
num2 = float(num_2)
print( num1 , type(num1) )
print( num2 , type(num2) )
```

Bool The function convert from number to boolean converting

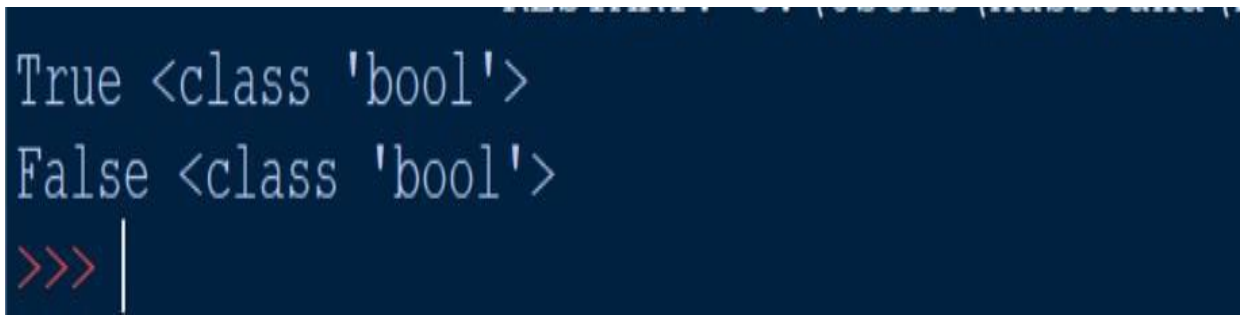
valid values to logical values

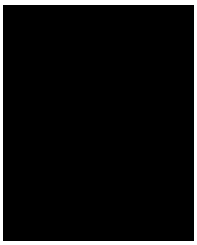
That was the following

num1 = 1
num2 = 0
bool1 = bool(num1)
bool2 = bool(num2)
print(bool1 , type(bool1))
print(bool2 , type(bool2))



Implementation





Ord Function ASCII convert letter to ord the character in a function ascii arrive to

That was the following


```
c = 'A'
i = ord(c)
print( i )
```

[redacted] Implementation

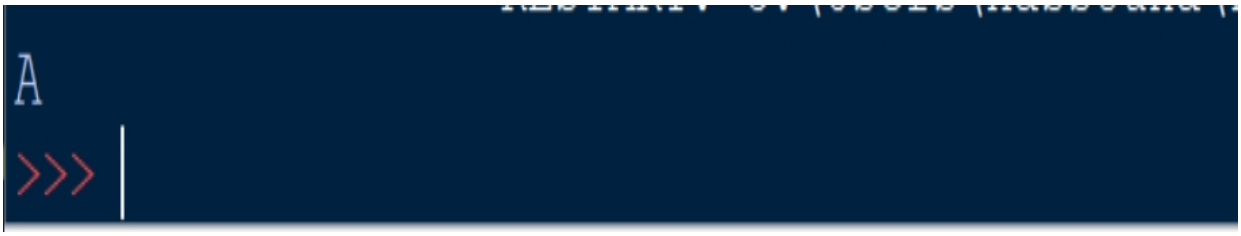
[redacted] Chr the function for a letter ASCII convert the chr by function

ascii get to the letter by putting

That was the following


```
i = 65
c = chr(i)
print( c )
```

Implementation



Range Function span variable to make a quick and fix list

range utilization

That was the following

The end of the number that does not reach it


```
r = range(5)
print( r[0], r[1], r[2], r[3], r[4] )
```

Implementation


Code

r = range(7)
print(r[0], r[1], r[2], r[3], r[4], r[5], r[6])


Implementation

[redacted] The end he never reaches Determine the start then What follows

[redacted]

[redacted]

[redacted]

[redacted] has been done

[redacted]

r = range(1 , 8)
print(r[0], r[1], r[2], r[3], r[4], r[5], r[6])

[redacted] Implementation


```
1 2 3 4 5 6 7
>>>
```

That the next step was used and determined, whether positive or negative

r = range(1,11 , 2)

print(r[0], r[1], r[2], r[3], r[4])

Implementation

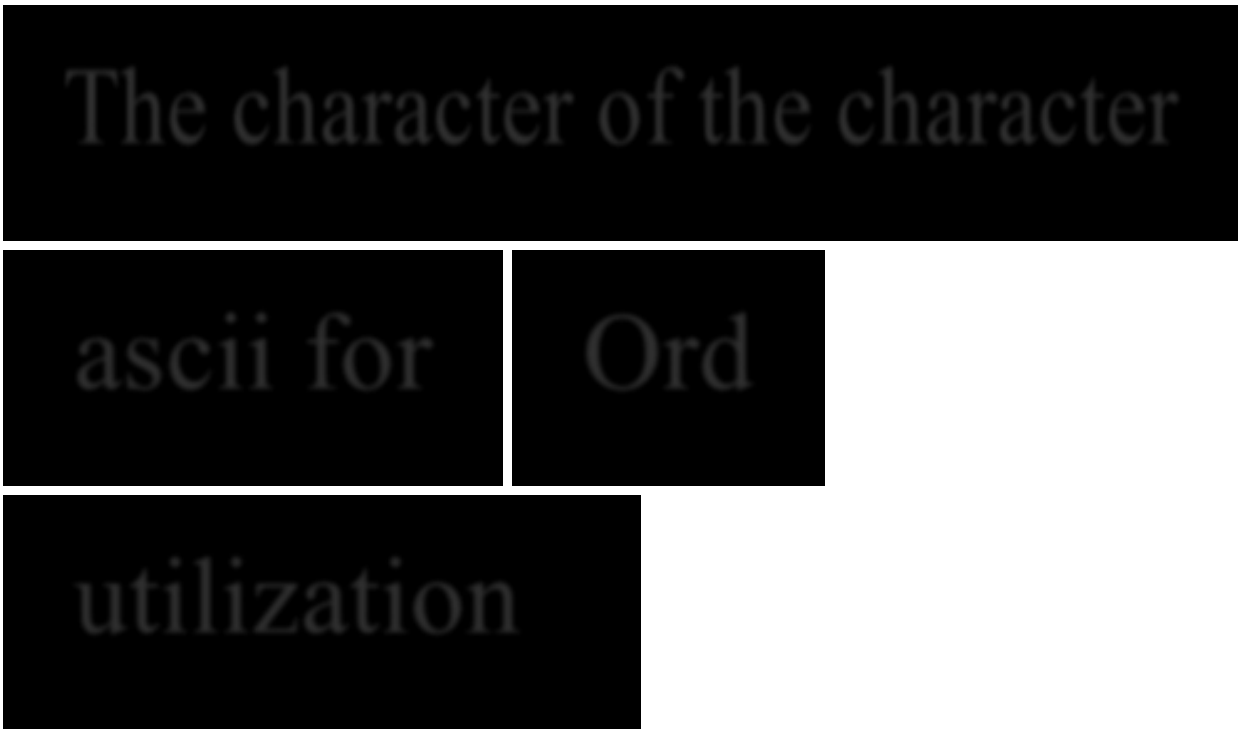

```
1 3 5 7 9
>>>
```

Code


```
r = range(2,21 , 3)
print( r[0], r[1], r[2], r[3], r[4], r[5], r[6] )
```

Implementation

The character of the character ascii for Ord utilization



What follows has been done ascii for the character from chr

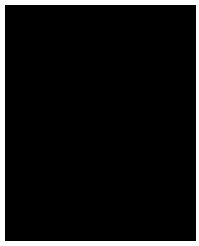
Code


```
r = range( ord('A'), ord('Z')+1 )
chr(r[0]), print( chr(r[1]), chr(r[2]), chr(r[3]),
chr(r[4]), chr(r[5]) )
```

Implementation

r = range(6, 1, -1)
print(r[0], r[1], r[2], r[3], r[4])

Implementation

 Code

r = range(5, 0, -1)
print(r[0], r[1], r[2], r[3], r[4])

[redacted] Implementation

[redacted]

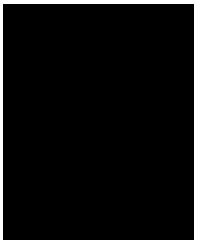
[redacted] Code

[redacted]

r = range(10, 0, -2)
print(r[0], r[1], r[2], r[3], r[4])

Implementation

[redacted] [redacted]



Input Function Receipt from the user in the following, the program waits for you to enter a value in the input function and

notice even if you enter a number

This value is textual

name = input("Enter your name:")
print('Hello ' + name)

Code

Code

name = input('Enter any name:')
print(type(name))

Implementation 1

Implementation 1

```
Enter any name:Amr
<class 'str'>
>>>
```

Implementation 2

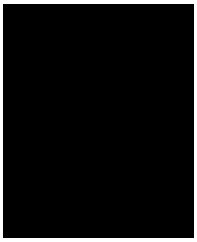

```
Enter any name:123
<class 'str'>
>>>
```

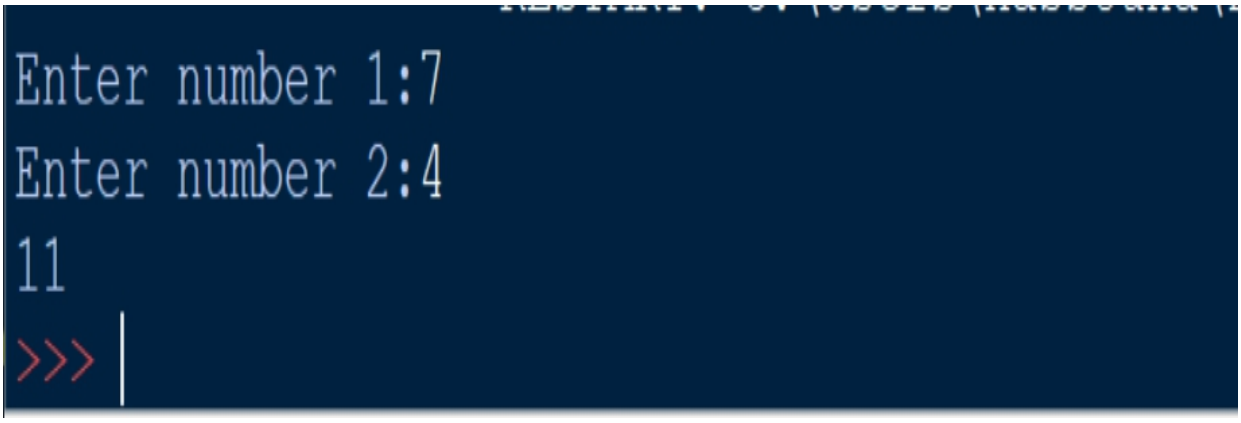
To be amenable int text number conversion using what follows

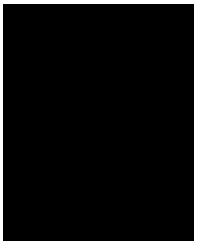
has been done to perform mathematical operations

The code


```
num1 = int( input('Enter number 1:') )
num2 = int( input('Enter number 2:') )
result = num1 + num2
print( result )
```







Randint and function import import and random -below is the randomly drawn module that will be imported modules, and that

function and so on for the rest of the import With the word to her max and the min Produces a random number as specified by the randint

The code


```
import random
num = random.randint(1,10); print(num)
num = random.randint(1,10); print(num)
```

num = random.randint(1,10); print(num)
num = random.randint(1,10); print(num)
num = random.randint(1,10); print(num)

Implementation 1

Implementation 2

Implementation 2

The code

The code

```
from random import randint
num = randint(10,20); print(num)
num = randint(20,30); print(num)
num = randint(30,40); print(num)
num = randint(40,50); print(num)
num = randint(50,60); print(num)
num = randint(60,70); print(num)
```

[redacted] Implementation

[redacted] and, or, not with words Boolean operators true unless all of its parties True do not return and that False unless all of its parties

False or do not return

bool1 = True and True; print('true AND true =',bool1)

print('false AND true =',bool2)

print('true AND false

print('false AND false =',bool4)

bool2 = False and True; bool3 = True and False;

bool4 = False and False; bool5 = True and True and False and True; print('T&T&F&T =',bool5) bool6 = True and True and True and True; print('T&T&T&T =',bool6) print('==========')

print('true OR true =',bool7)

bool7 = True or True;
bool8 = False or True;

bool10 = False or False;
print('false OR true =',bool8) print('true OR false =',bool9)

print('false OR false =',bool10)

bool11 = False or False or False or False; print('F|F|F|F =',bool11)

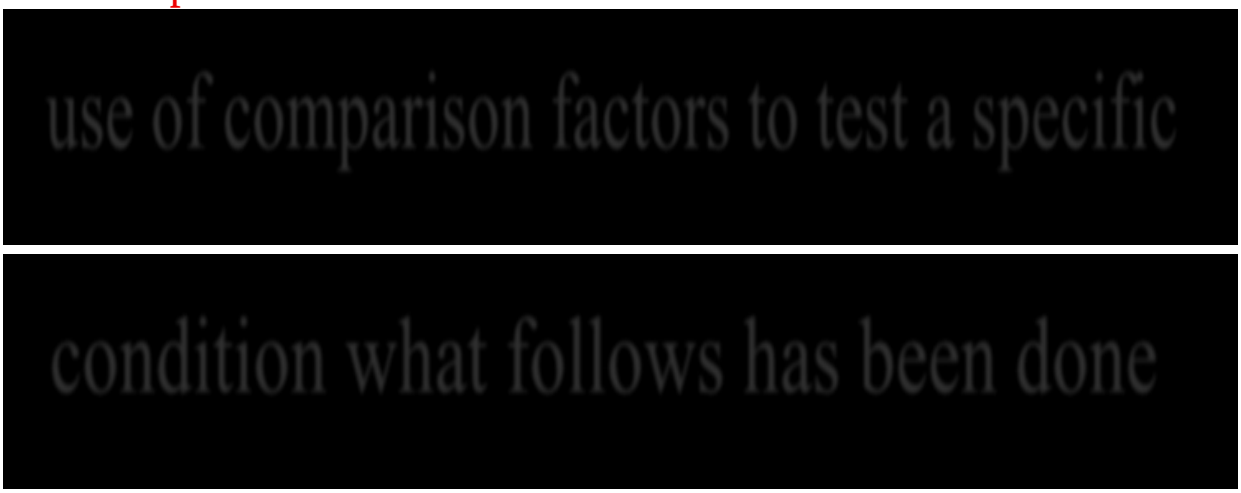
bool12 = False or False or True or False; print('F|F|T|F =',bool12) bool13 = True or True or True or True; print('T|T|T|T =',bool13) print('===============')

bool14 = not True; print('NOT true =',bool14)

Implementation

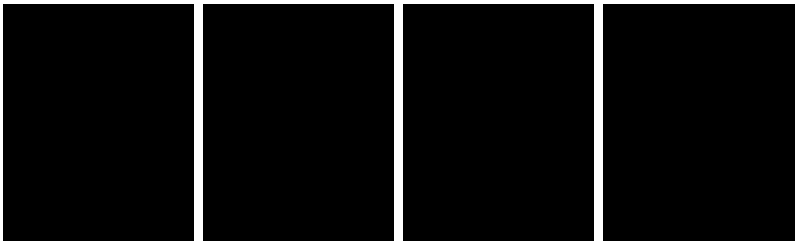
```
true AND true   = True
false AND true  = False
true AND false  = False
false AND false = False
T&T&F&T = False
T&T&T&T = True
==============
true OR true    = True
false OR true   = True
true OR false   = True
false OR false  = False
F|F|F|F = False
F|F|T|F = True
T|T|T|T = True
==============
NOT true   = False
NOT false = True
>>>
```

Comparison factors use of comparison factors to test a specific condition what follows has been done



x , y = 7 , 9
b1 = x>y; print(b1) b2 = x<y; print(b2) b3 = x>=y; print(b3) b4 = x<=y; print(b4) b5 = x==y; print(b5) b6 = x!=y; print(b6)

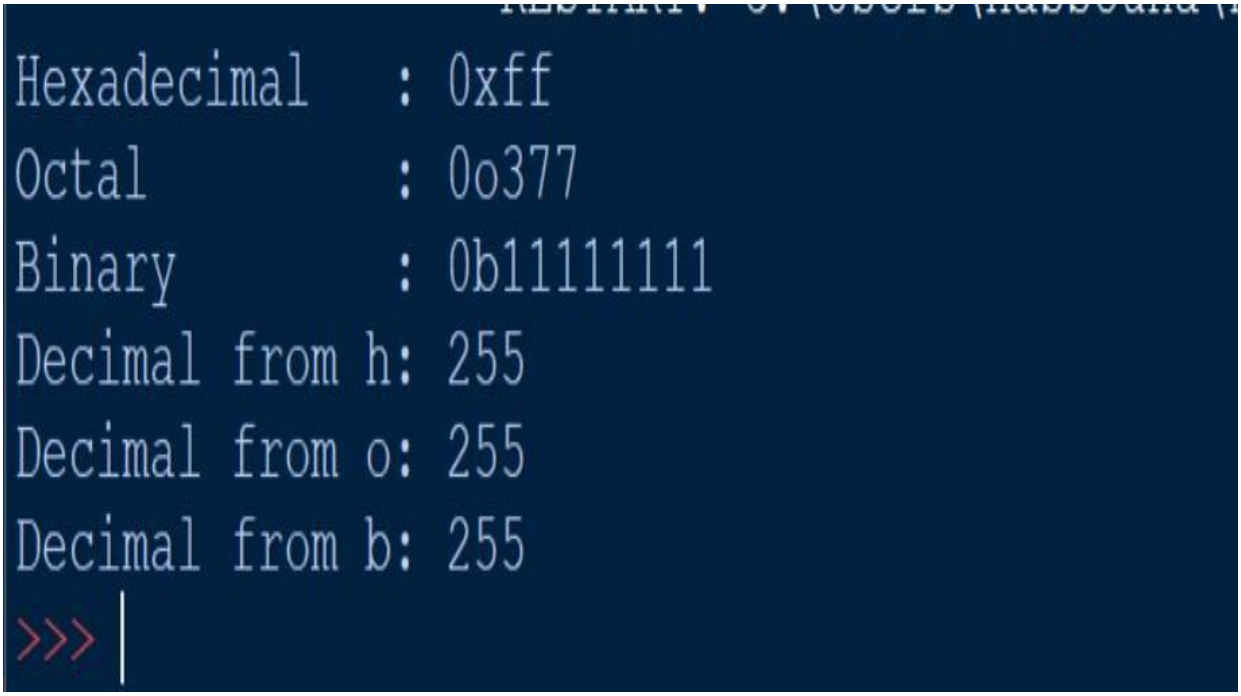
Implementation

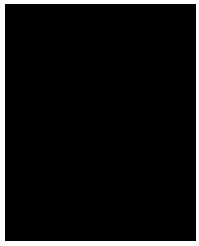


```
False
True
False
True
False
True
>>>
```

Transformations of numerical systems transfer between numerical systems in more than one way what follows has been done

num_d = 255
num_h = 0xff
num_o = 0o377
num_b = 0b11111111
print('Hexadecimal :', hex(num_d)) print('Octal :', oct(num_d))
print('Binary :', bin(num_d)) print('Decimal from h:', int(num_h))
print('Decimal from o:', int(num_o)) print('Decimal from b:', int(num_b))



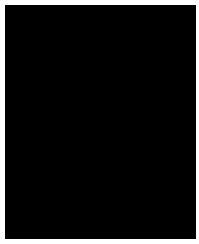


The code


```
num_d = 255
num_h = 0xff
num_o = 0o377

num_b = 0b11111111 :', format(num_d,'x') ) print( 'Hexadecimal

print( 'Octal :', format(num_d,'o') ) print( 'Binary :',
print( 'Decimal from h:',
print( 'Decimal from o:',
print( 'Decimal from b:',
format(num_d,'b') ) format(num_h,'d') ) format(num_o,'d') )
format(num_b,'d') )
```

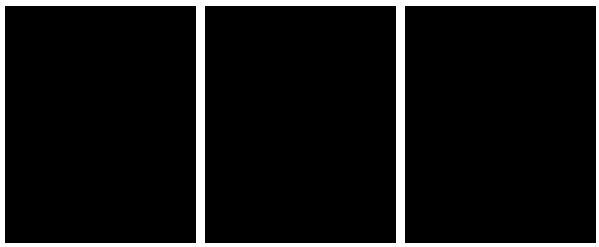
Implementation

```
Hexadecimal   : ff
Octal         : 377
Binary        : 11111111
Decimal from h: 255
Decimal from o: 255
Decimal from b: 255
>>>
```



The code


```
print( int('11111111',2) )
print( int('377',8) )
print( int('255',10) )
print( int('ff',16) )
```



Chopping text chopping the text with ease as if we are dealing

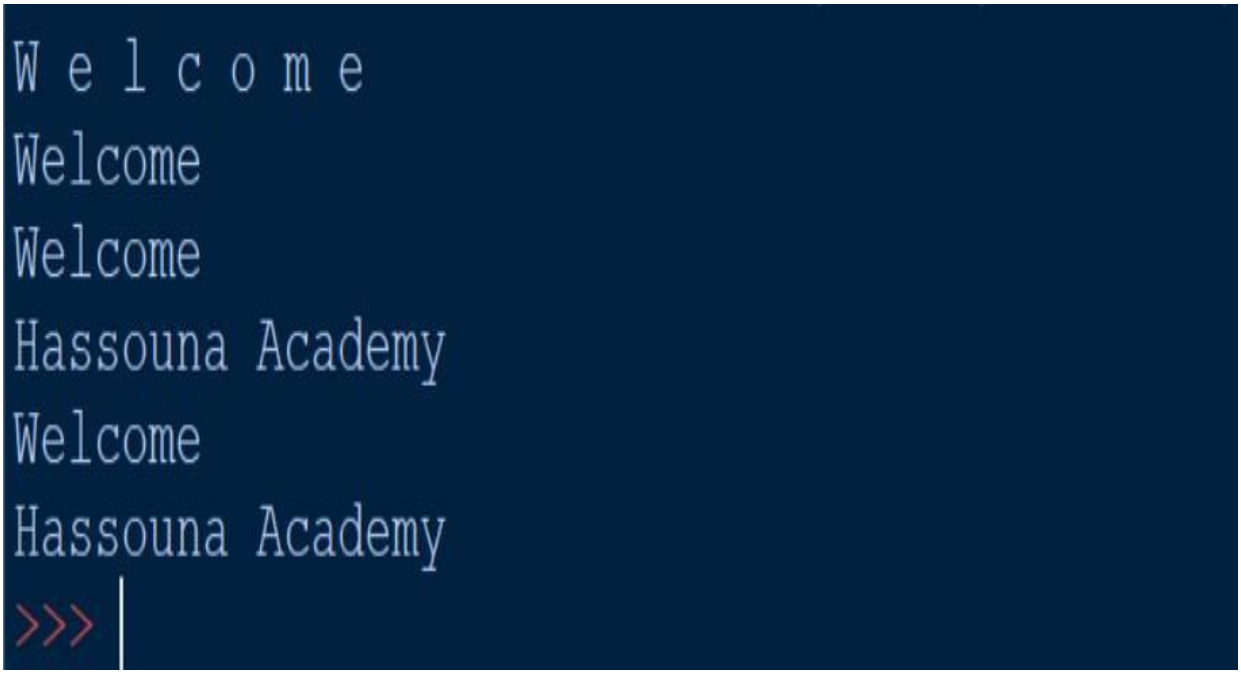
with what follows has been done just and we define what we

want for it with the range between the two points:

Code List


```
my = 'Welcome to Hassouna Academy'
print( my[0], my[1], my[2], my[3], my[4], my[5], my[6] ) print( my[:7] )
print( my[0:7] )
print( my[11:] )
print( my[:-len(my)+7] )
print( my[11:len(my)] )
```

Implementation



 To separate text into a list split use the function what follows

has been done


```
str_names = 'Ahmed;Adel;Amr;Ali;Omar;Haitham'
list_names = str_names.split(';')
print( str_names )
print( list_names )
```

[redacted] Implementation


```
Ahmed;Adel;Amr;Ali;Omar;Haitham
['Ahmed', 'Adel', 'Amr', 'Ali', 'Omar', 'Haitham']
>>>
```

[redacted] Connect the text list connect text content, whether text or

what follows has been done

```
str1 = 'Hello'
str2 = '-'.join(str1)
print( str1 )
print( str2 )
```

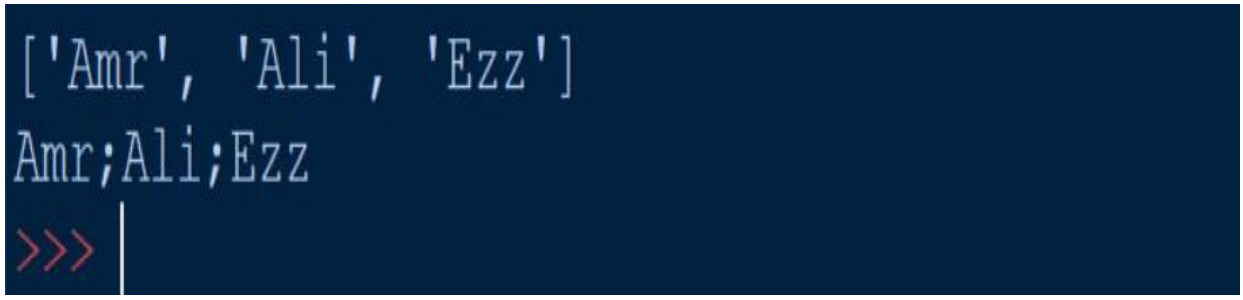
Implementation

The code


```
list_names = ['Amr','Ali','Ezz']
str_names = ';'.join(list_names)
```

print(list_names)
print(str_names)

Implementation



list_names = ['Amr','Ali','Ezz','Ehab']
str_names = '\n'.join(list_names)
print(list_names)
print(str_names)

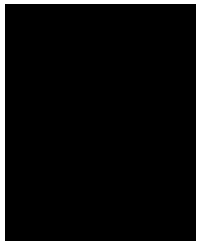
[redacted] Implementation


```
['Amr', 'Ali', 'Ezz', 'Ehab']
Amr
Ali
Ezz
Ehab
>>>
```

[redacted] Text formatting for text formatting use of emoticons what follows has been done


```
name = 'Amr'
my = 'Hello %s' % name
print( my )
```

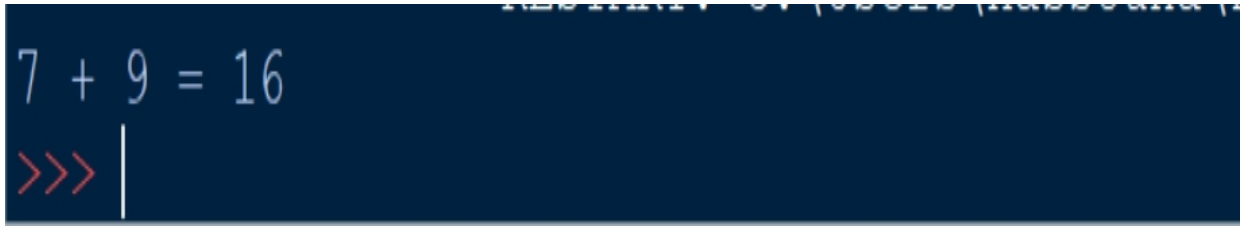

Implementation



The code


```
num1 = 7
num2 = 9
my = '%d + %d = %d' %(num1, num2, num1+num2)
print( my )
```

Implementation



```
my = '65 is ASCII for %c' % 'A'
print( my )
```

[redacted] Implementation

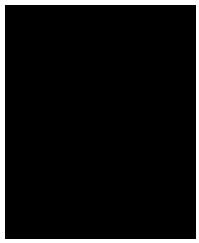
The code

[redacted] **my = 'Character %c'**
my += '\nString %s'

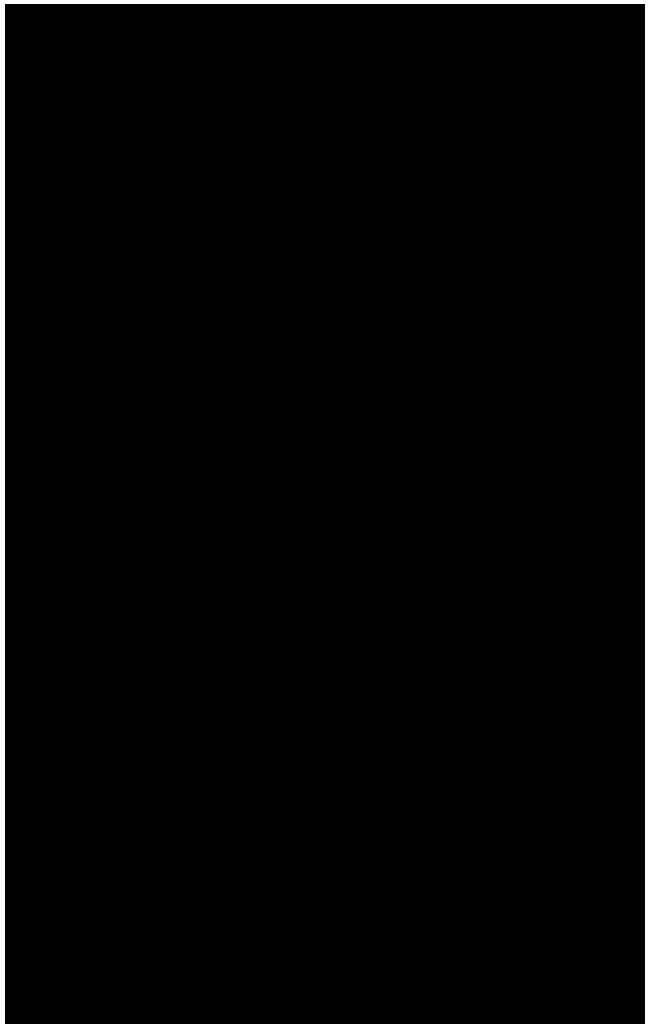
my += '\nDecimal %d'
my += '\nInteger %i'
my += '\nexponent %e'
my += '\nExponent %E'
my += '\nFloat %f'

my += '\nFloat %0.2f'

my += '\nNumber %g,%g'



% 'A'
% 'Hi'
% 55.99
% 77
% 33
% 33

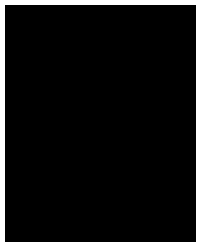


% 99.77

my += '\nHexadecimal %x' % 65

print(my)

```
Character A
String Hi
Decimal 55
Integer 77
exponent 3.300000e+01
Exponent 3.300000E+01
Float 99.770000
Float 99.77
Number 3.7,99
Octal 101
Hexadecimal 41
>>>
```



Use another type of initialization, where it is written

Use another type of initialization, where it is written What

follows has been done Names are enclosed in brackets {} in the

text and then specify any values for them to be displayed

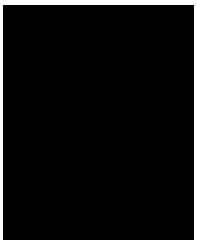
We also wish, thanks to God

The code


```
name = 'Ahmed'
say_hello = 'Hello {my}'
```

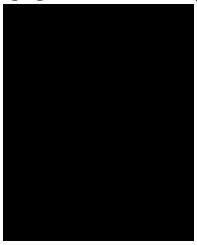
my_format = say_hello.format(my=name)
print(my_format)

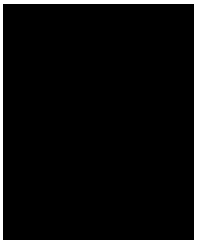
Implementation



The code

my_format = '{n1} + {n2} = {r}'.format(n1=7, n2=3, r=7+3) print(my_format)





ABC uppercase and lowercase transformations, lower and upper

Text conversion what follows has been done

The code

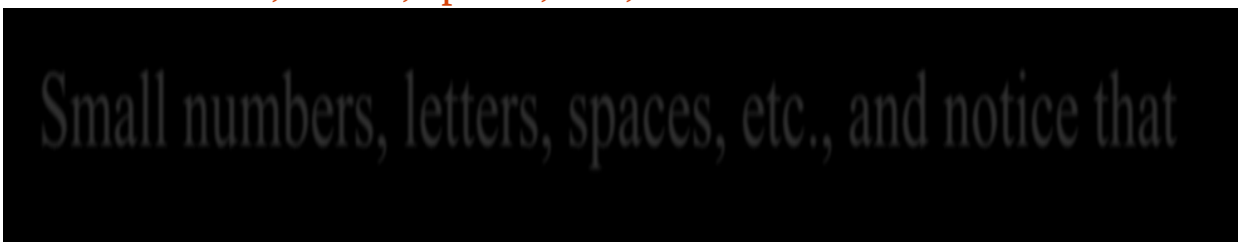

```
str1 = 'HELLO'
str2 = 'welcome'
print( str1.lower() )
print( str2.upper() )
```


Implementation

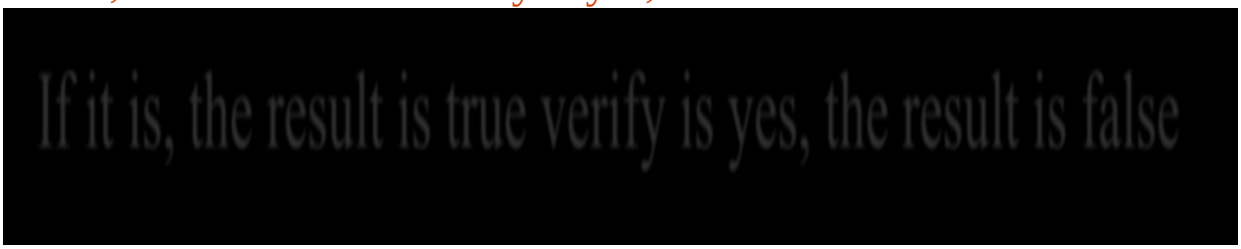

Text check to find out whether it is uppercase or text

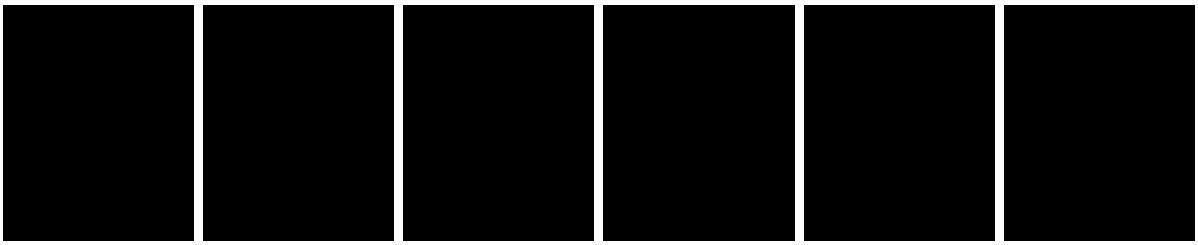
verification what follows has been done

Small numbers, letters, spaces, etc., and notice that



If it is, the result is true verify is yes, the result is false






```
print( 'HELLO'.isupper() )
print( 'hello'.islower() )
print( 'HEllo'.isalpha() )
print( 'ABC45'.isalnum() )
print( '12345'.isdigit() )
```

```
print( ' '.isspace() )
print( '1 AB@'.isprintable() )
print('==========')
print( 'HeLLO'.isupper() )
print( 'hEllo'.islower() )
print( 'HE7lo'.isalpha() )
print( 'AB@45'.isalnum() )
print( '12A45'.isdigit() )
print( ' . '.isspace() )
print( '\n'.isprintable() )
```

Implementation

```
True
True
True
True
True
True
True
=========
False
False
False
False
False
False
False
>>>
```

Text search keyword by search index arrive to what follows has been done


```
my = 'Hello Amr and Welcome Back Amr'
indexFind1 = my.find('amr')
indexFind2 = my.find('Amr')
print( indexFind1 )
print( indexFind2 )
```

Implementation

```
-1
6
>>>
```


The code


```
my = 'Hello Amr and Welcome Back Amr'
i = my.find('Welcome')
print( my[i:] )
```


Implementation


Text replacement replace text with other text what follows has been done

The code


```
my1 = 'Hello Amr and Welcome Back Amr'
my2 = my1.replace('Amr','Adel')
print( my1 )
print( my2 )
```

[redacted] Implementation

[redacted]

Decision making - the if statement the implementation of the

code depends on a specific condition, if it is the result what follows has been done not implemented false runs and if b

True the condition

x = 5

if x==5: print('OK')

Implementation

The code

x = 7
if x>5:print('OK1');print('OK2');print('OK3')

Implementation

x = 7
if x<=7:

print('OK1')
print('OK2')
print('OK3')

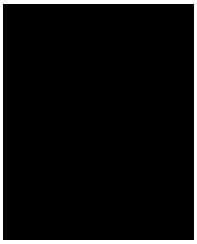
Implementation

The code

num = int(input('Enter any number:'))

```
if num<0:
print('Negative Number')
else:
print('Positive Number')
```

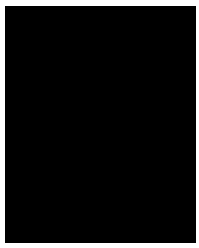
Implementation 1



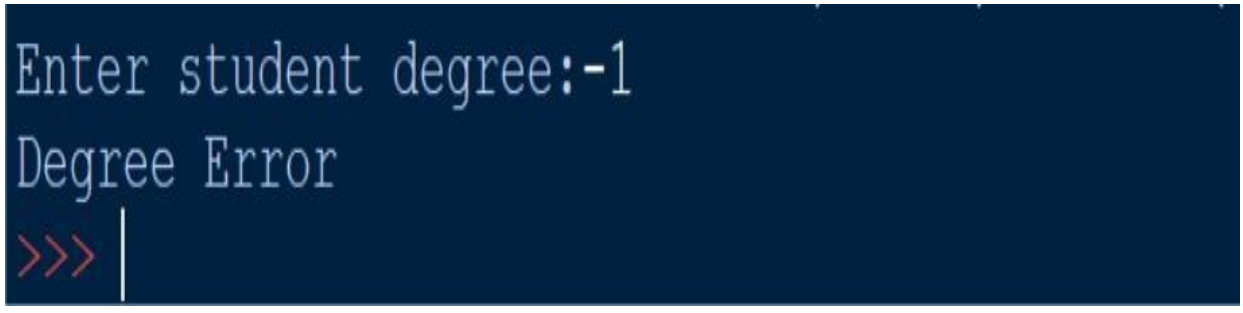
Implementation 2


```
degree = int( input('Enter student degree:') )
```

```
if degree<0 or degree>100:
print('Degree Error')

elif degree<50:
print('F')
elif degree<60:
print('E')
elif degree<70:
print('D')
elif degree<80:
print('C')
elif degree<90:
print('B')
else:
print('A')
```



Implementation 1



Implementation 2


```
Enter student degree:0
F
>>>
```

Implementation 3


```
Enter student degree:40
F
>>>
```

```
Enter student degree:70
C
>>>
```

Implementation 5


```
Enter student degree:80
B
>>>
```

Implementation 6

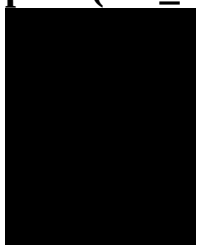
```
Enter student degree:95
A
>>>
```


Triple Conditional Expression operator if on one line for the shortcut utilization what follows has been done

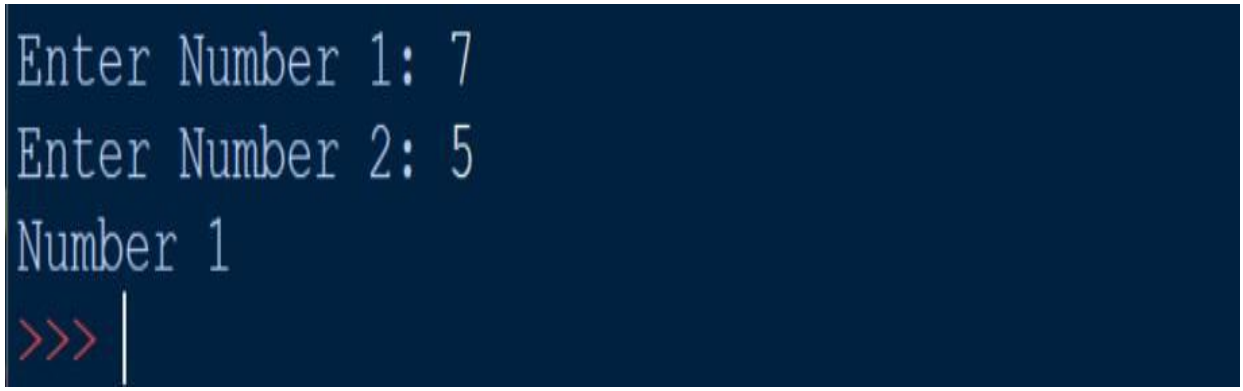
The code

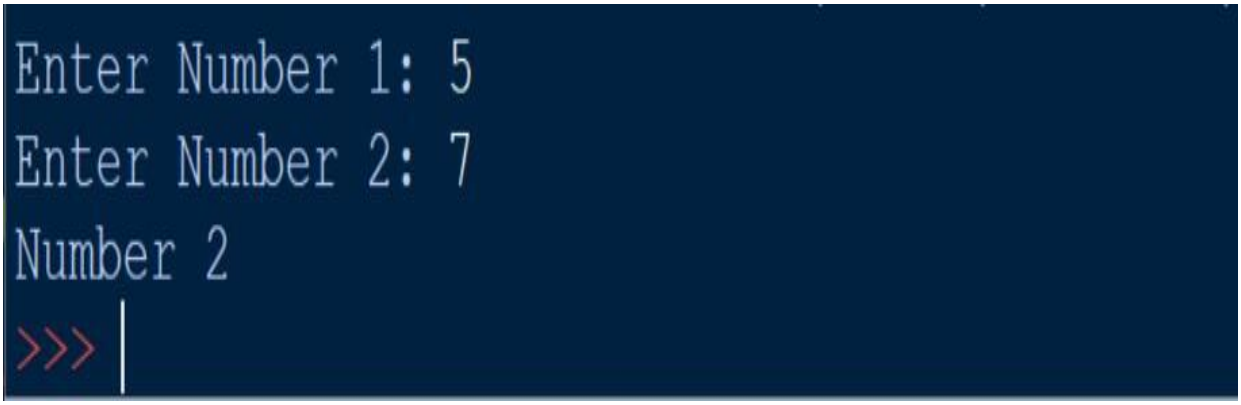

```
num1 = int( input('Enter Number 1: ') )
num2 = int( input('Enter Number 2: ') )
```

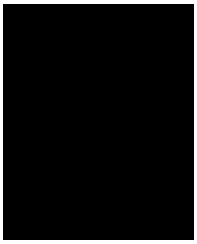
str_big = 'Number 1' if num1>num2 else 'Number 2'
print(str_big)



Implementation 1







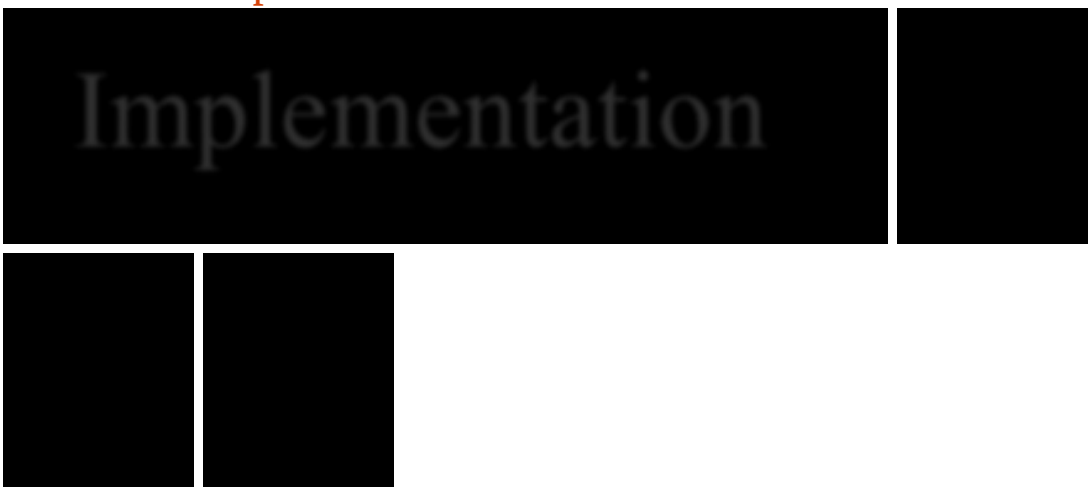
For Sentence loops of iteration in the following, iteration loops

were used to use code duplication and perform multiple operations with few lines of code to save time and effort

The code

for x in (1,2,3,4,5): print(x)

Implementation



The code

for x in [10,20,30,40,50]:

print(x)

Implementation

The code

for x in range(2,11 , 2):

print(x)

Implementation

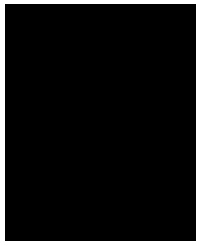
The code


```
for x in range(1,6):
if x != 4:
print( x )
```

Implementation

The code

for x in range(5, 0, -1):
print(x)



The code


```
alpha = ''
for x in range( ord('A'), ord('Z')+1 ):
alpha += chr(x)
if x<ord('Z'): alpha += ', '
```

print(alpha)

Implementation

The code


```
alpha = ''
for x in range( ord('Z'), ord('A')-1, -1 ):
alpha += chr(x)
if x>ord('A'): alpha += ', '
```

print(alpha)

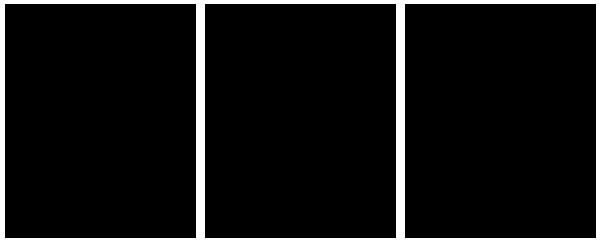
```
Z, Y, X, W, V, U, T, S, R, Q, P, O, N, M, L,
K, J, I, H, G, F, E, D, C, B, A
>>>
```

The code

names = ['Ahmed','Adel','Amr','Omar','Ali']
for x in range(len(names)):
print('Hello ' + names[x])

Implementation

```
Hello Ahmed
Hello Adel
Hello Amr
Hello Omar
Hello Ali
>>>
```

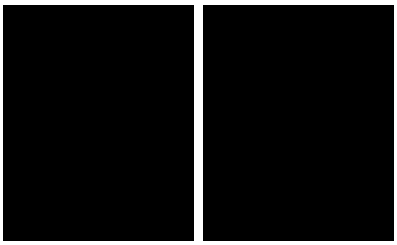

The code

names = ['Ahmed','Adel','Amr','Omar','Ali']
for name in names:

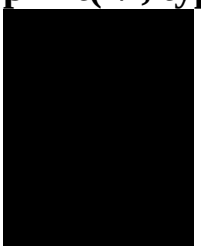

print('Hello ' + name)

Implementation

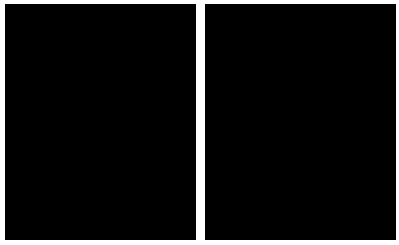
```
Hello Ahmed
Hello Adel
Hello Amr
Hello Omar
Hello Ali
>>>
```

 The code

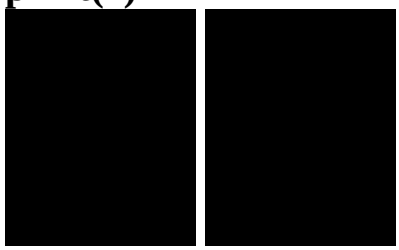
my_list = [3, 'A', True, 5.7]
for v in my_list:
print(v , type(v))

 Implementation

```
3 <class 'int'>
A <class 'str'>
True <class 'bool'>
5.7 <class 'float'>
>>>
```

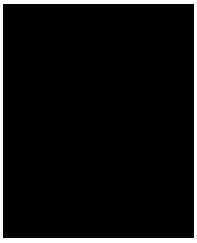

The code

emp = { 'name':'Adel', 'city':'Giza', 'salary':3000 } for x in emp:
print(x)

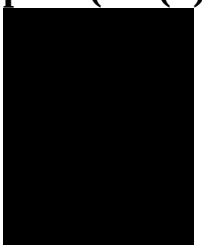

Implementation

emp = { 'name':'Adel', 'city':'Giza', 'salary':3000 } for x in emp:
print(emp[x])

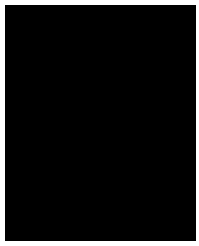
Implementation


The code

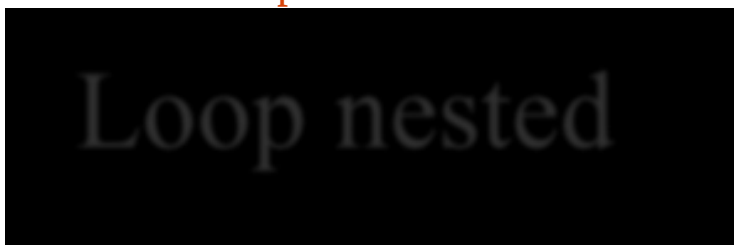
emp = { 'name':'Ahmed', 'city':'Giza', 'salary':3000 } for k,v in emp.items(): print(str(k) + ':' , v)


Implementation

```
name: Ahmed
city: Giza
salary: 3000
>>>
```



Loop nested



Use overlapping redundancy that needs a pinnacle what follows has been done focus and understanding to be simple for you, and

luck that cross-repetition can or more loop inside loop to be

the code

family1 = ['Ahmed','Adel','Amr'] family2 = ['Ehab','Mahmoud','Ezz']
family3 = ['Sarah','Hajer','Rehab']

home1 = [family1 , family2 , family3]

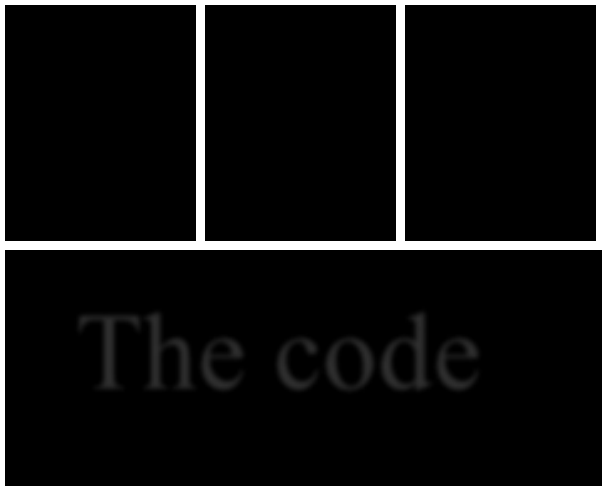
for x in range(len(home1)):
print('Family:', x+1)
for y in range(len(home1[x])):

print(' Name', y+1, 'is:', home1[x][y])

Implementation

Implementation

```
Family: 1
    Name 1 is: Ahmed
    Name 2 is: Adel
    Name 3 is: Amr
Family: 2
    Name 1 is: Ehab
    Name 2 is: Mahmoud
    Name 3 is: Ezz
Family: 3
    Name 1 is: Sarah
    Name 2 is: Hajer
    Name 3 is: Rehab
>>>
```




```
family1 = ['Adel','Amr'] family2 = ['Ehab','Ezz'] family3 =
['Sarah','Hajer']

family4 = ['Ezzat','Foaad']
family5 = ['Abdelrahman','Abdelkareem'] family6 = ['Ali','Akl']

home1 = [ family1 , family2 , family3 ] home2 = [ family4 , family5 ,
family6 ]
homes = [ home1 , home2 ]

for x in range( len(homes) ):
print( 'Home', x+1 )
for i in range( len(homes[x]) ):

print( ' Family', i+1 )
for y in range( len(homes[x][i]) ):
print( ' Name', y+1, 'is:', homes[x][i][y] )
```

Implementation

Implementation

```
Home 1
   Family 1
      Name 1 is: Adel
      Name 2 is: Amr
   Family 2
      Name 1 is: Ehab
      Name 2 is: Ezz
   Family 3
      Name 1 is: Sarah
      Name 2 is: Hajer
Home 2
   Family 1
      Name 1 is: Ezzat
      Name 2 is: Foaad
   Family 2
      Name 1 is: Abdelrahman
      Name 2 is: Abdelkareem
   Family 3
      Name 1 is: Ali
      Name 2 is: Akl
>>>
```

With more than one variable for create a repeat list with the list

enumerate utilization what follows has been done for two

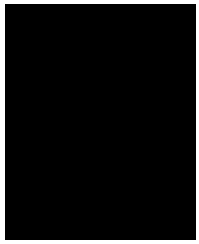
variables were dealt with within the iteration

The code

for i, name in enumerate(['amr','ali','ezz']):
print(i, name)

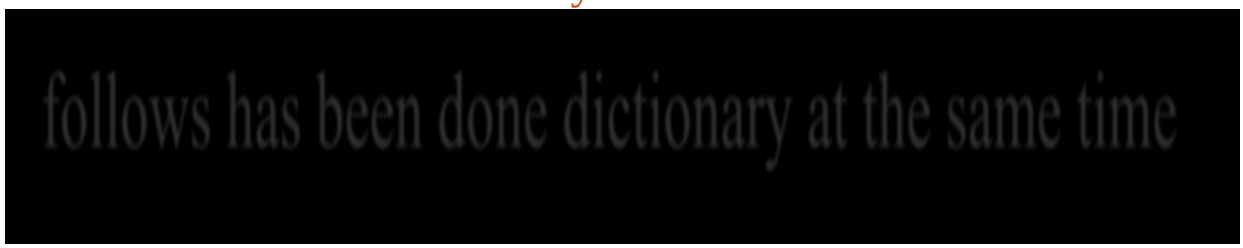
Implementation

```
0 amr
1 ali
2 ezz
>>>
```



A counter variable was used and also switches were used what

follows has been done dictionary at the same time

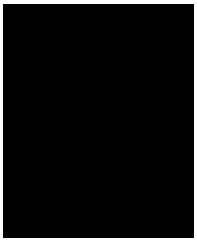


The code

person1 = { 'name':'Amr', 'salary':5000 }
person2 = { 'name':'Ali', 'salary':4000 }
person3 = { 'name':'Ezz', 'salary':3000 }

```
persons = [ person1, person2, person3 ]
for x in range( len(persons) ):

print( 'Person', x+1 )
for index, (k, v) in enumerate(persons[x].items()): print( ' ', index+1 , ':', k, v )
```

```
Person 1
    1 : name Amr
    2 : salary 5000
Person 2
    1 : name Ali
    2 : salary 4000
Person 3
    1 : name Ezz
    2 : salary 3000
>>>
```



While sentence loops of iteration the following iterations were

While sentence loops of iteration the following iterations were

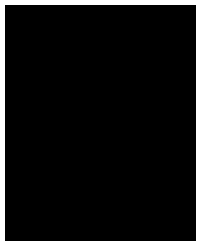
used to use code duplication and it was worked multiple operations with few lines of code to save time and effort

The code


```
x = 1
while x <= 5:
print( x )
```

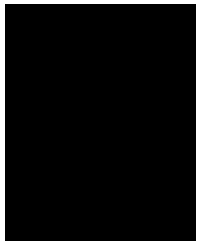
```
x +=1
```

Implementation

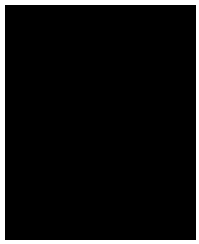


The code


```
x = 5
while x > 0:
print( x )
x -=1
```



Implementation



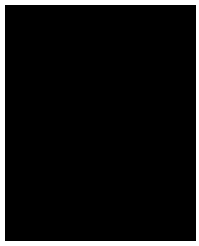
The code

x = 2

while x <= 10:

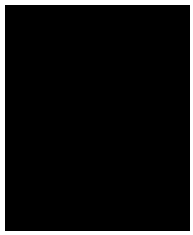
print(x)
x +=2

Implementation

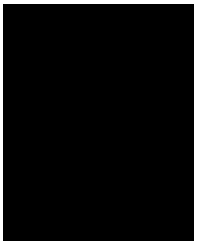


The code


```
x = 1
while x <= 10:
print( x )
```



```
x +=2
```

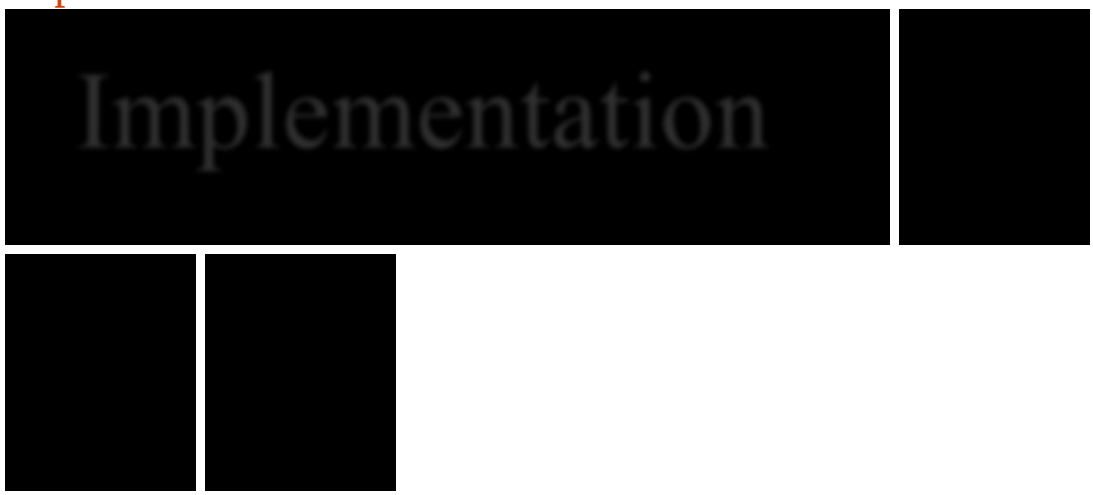


The code

x = 1
while x <= 10:
print(x)
x +=2
else:

print('X After Loop Is:', x)

Implementation



The code

```
x = 1
while x < 1:
print( x )
x +=2
else:
```

```
print('Condition is False')
```

Implementation 

 The code


```
my_list = [7,'A',9.9,False]
x = 0
```

while x < len(my_list):

print(my_list[x] , type(my_list[x]))
x += 1

[redacted] Implementation

[redacted]

[redacted]

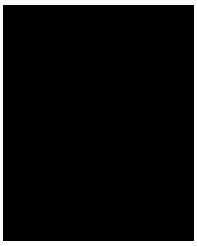
```
7 <class 'int'>
A <class 'str'>
9.9 <class 'float'>
False <class 'bool'>
>>>
```

The code

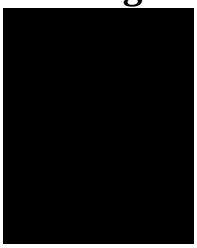
emp = { 'name':'Adel', 'city':'Giza', 'salary':3000 } my_keys = list(emp.keys())
x = 0
while x < len(emp):

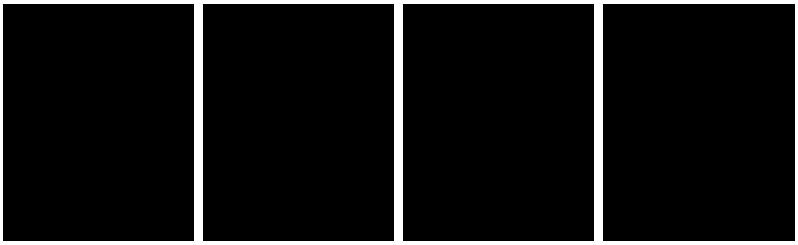
print(emp[my_keys[x]])
x += 1

Implementation

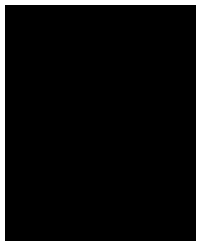

again = 'y'

while again=='y':


Implementation



```
Enter your name:Ahmed
Hello Ahmed
Again(y/n)?:y
Enter your name:Amr
Hello Amr
Again(y/n)?:y
Enter your name:Adel
Hello Adel
Again(y/n)?:n
>>>
```

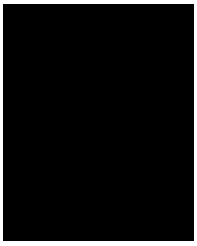


Infinite redundancy

Infinite

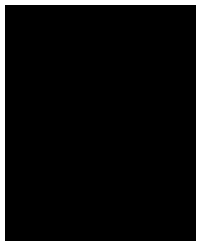
loop makes the program not stop and keep running


```
x = 1
while True:
print(x)
x+=1
```

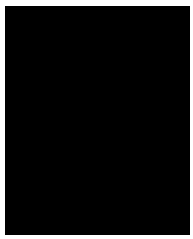


Implementation

```
5159
5160
5161
5162
5163
5164
5165
5166
```


69

name = input('Enter your name:') print('Hello ' + name)
again = input('Again(y/n)?:')



from itertools import count
for x in count():

[redacted] **print(x)**

[redacted]

[redacted] Implementation

[redacted]


Break

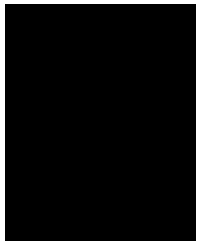
In what follows, the iteration has been permanently exit and

completely stopped using break

Code Sentence


```
for x in range(1,6):
if x == 4: break
print( x )
```

Implementation



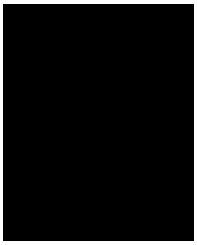
```
x = 1
while x <= 100:
if x>5:

break
print( x )
x += 1
```

Implementation

The code

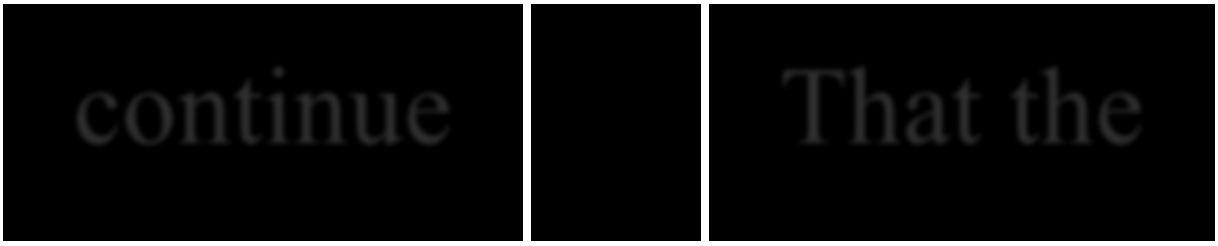

```
x = 1
while x <= 100:
if x>3: break
print( 'OK', x )
x += 1
```

Implementation



Continue in iterations – continue You ignore what is underneath

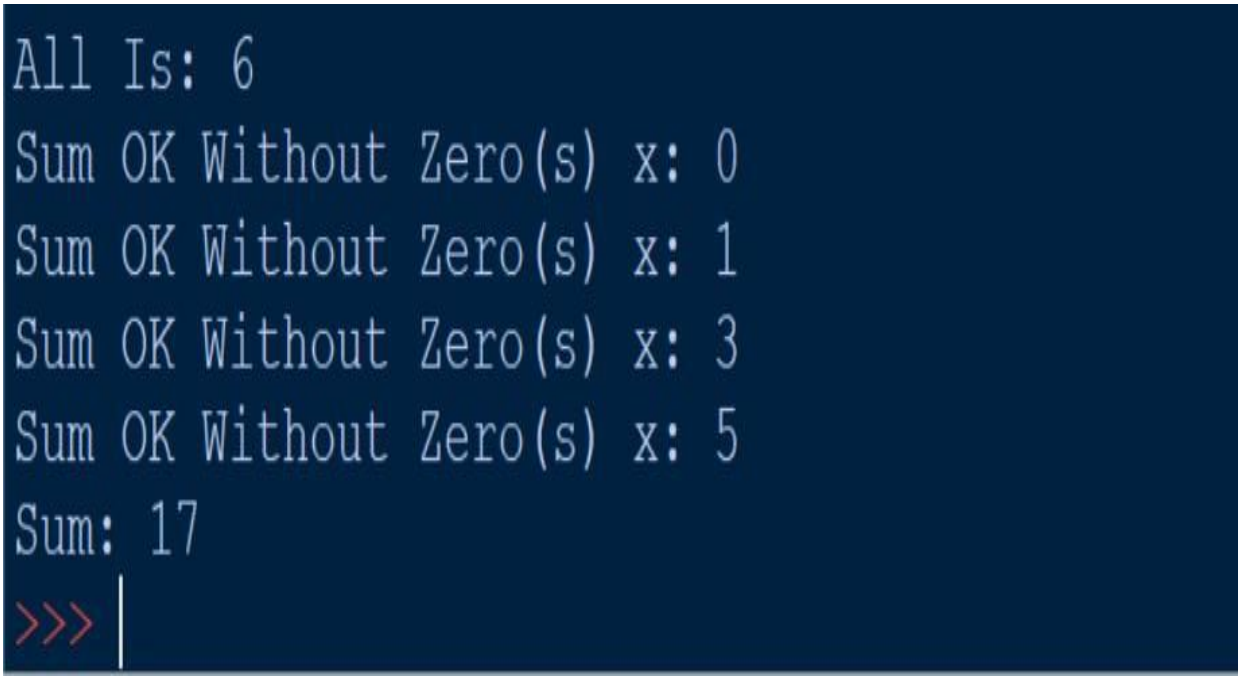
and go on and on continue That the Normal to repeat

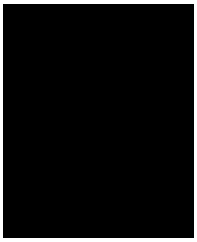


numbers = [5,2,0,3,0,7]
mysum = 0

```
print( 'All Is:', len(numbers) )
for x in range( len(numbers) ):

if numbers[x]==0: continue
mysum += numbers[x]
print('Sum OK Without Zero(s)','x:',x)

print( 'Sum:', mysum )
```

Implementation

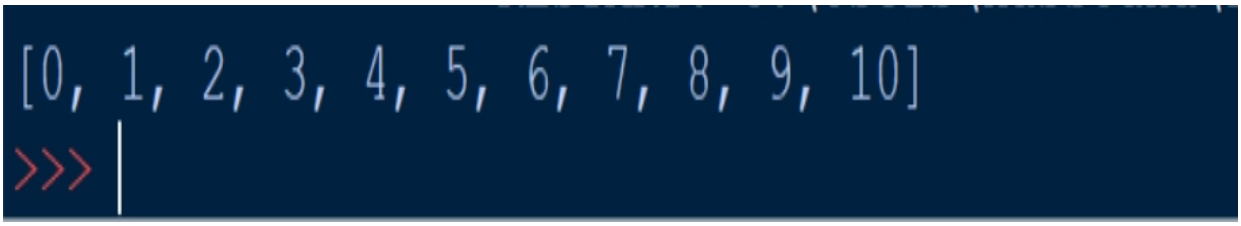




Create a list of repetitions create a list from nothing using repetition what follows has been done

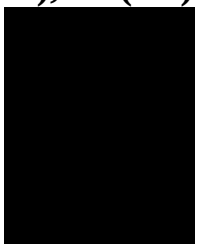

```
numbers = [ num for num in range(11)]
print( numbers )
```

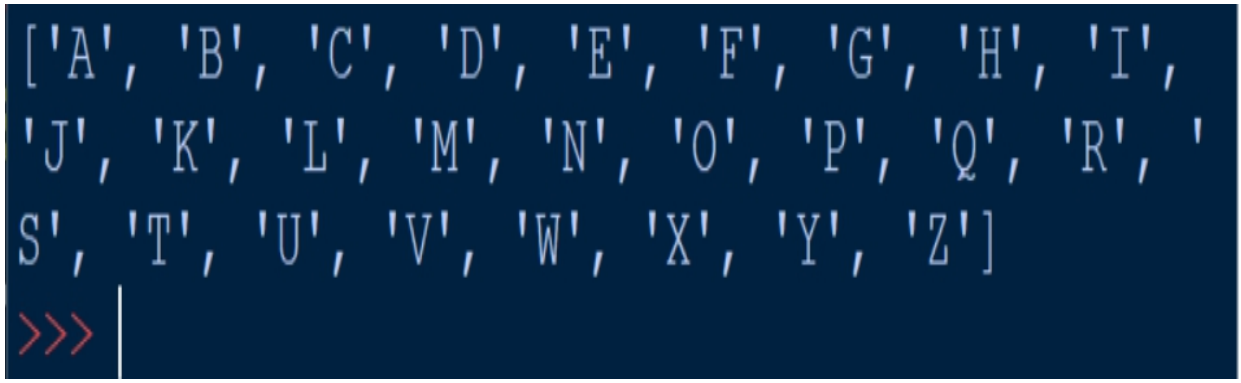

Implementation




The code

numbers = [chr(num) for num in range(ord('A'),ord('Z')+1)] print(numbers)

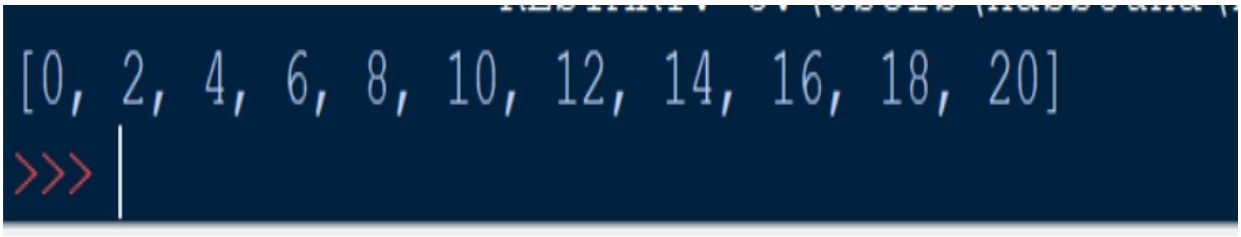


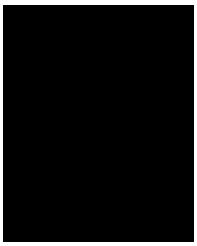


The code

numbers = [num for num in range(21) if num%2==0] print(numbers)

Implementation



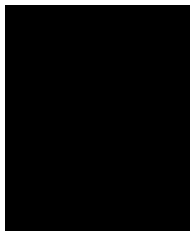


Print time and date for time now then datetime utilization what

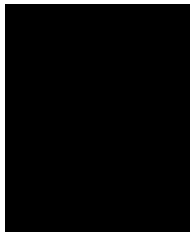
follows has been done and the current date, then we use them what we want, and notice that time can be made and a date with

the values that we want the code

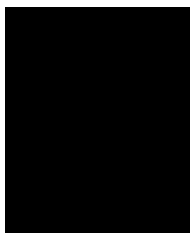
import datetime



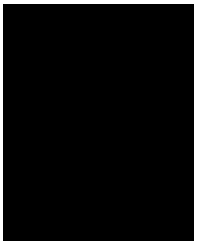
dt1 = datetime.datetime.now()



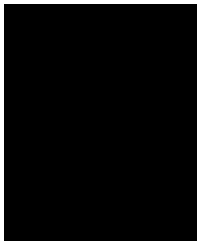
dt2 = datetime.datetime.now().date()
dt3 = datetime.datetime.now().time()



dt4 = datetime.date(2005,12,31)

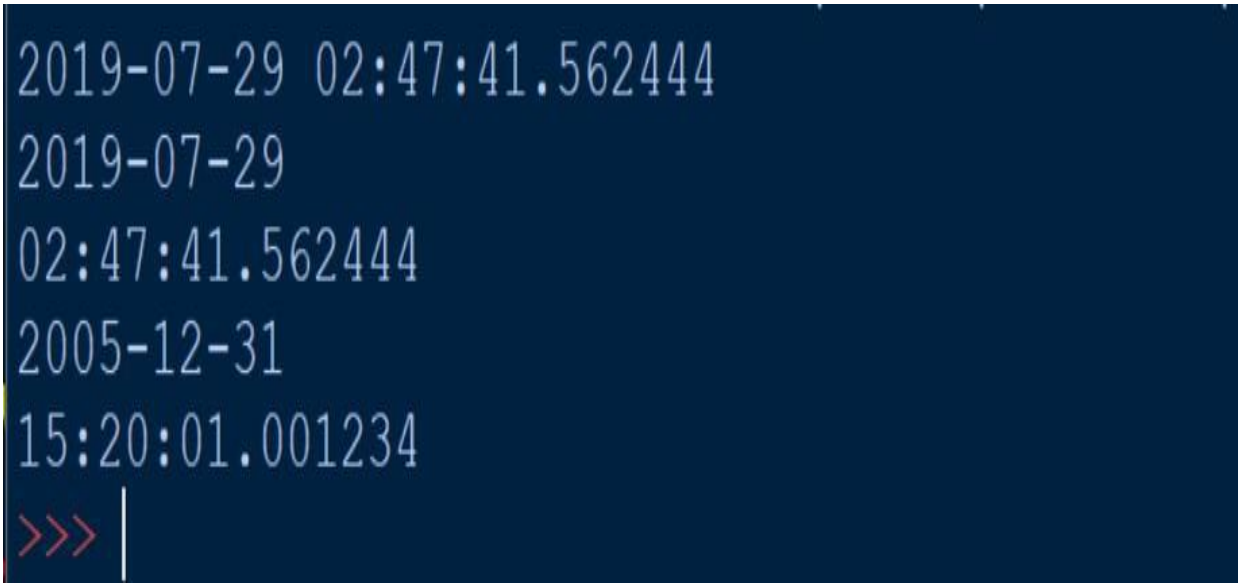


print(dt1)



print(dt2) print(dt3)

print(dt4) print(dt5)



Customize date and time customize time and date as we want what follows has been done

The code

import datetime

 **now = datetime.datetime.now()**

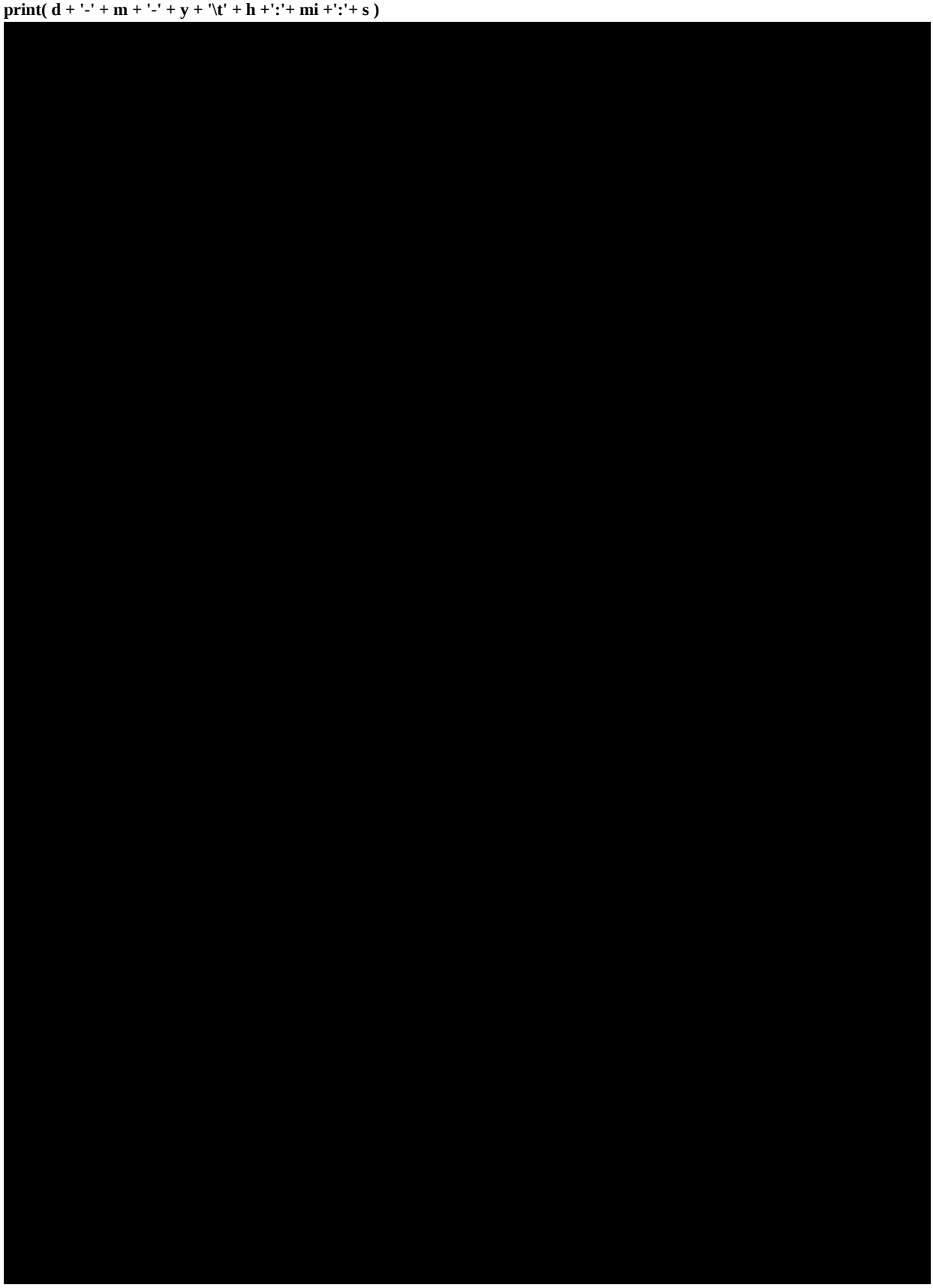
d = str(now.day)
m = str(now.month)
y = str(now.year)

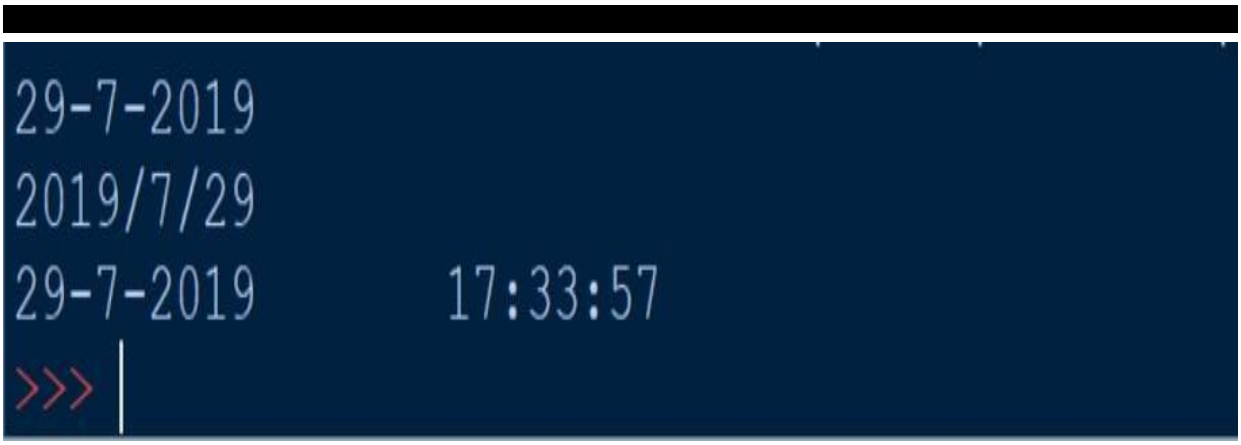
h = str(now.hour) mi = str(now.minute) s = str(now.second)

ms = str(now.microsecond)

print(d + '-' + m + '-' + y) print(y + '/' + m + '/' + d)

print(d + '-' + m + '-' + y + '\t' + h +':'+ mi +':'+ s)





Date and time initialization create a date and time configuration to facilitate our operations what follows has been done The code

import datetime

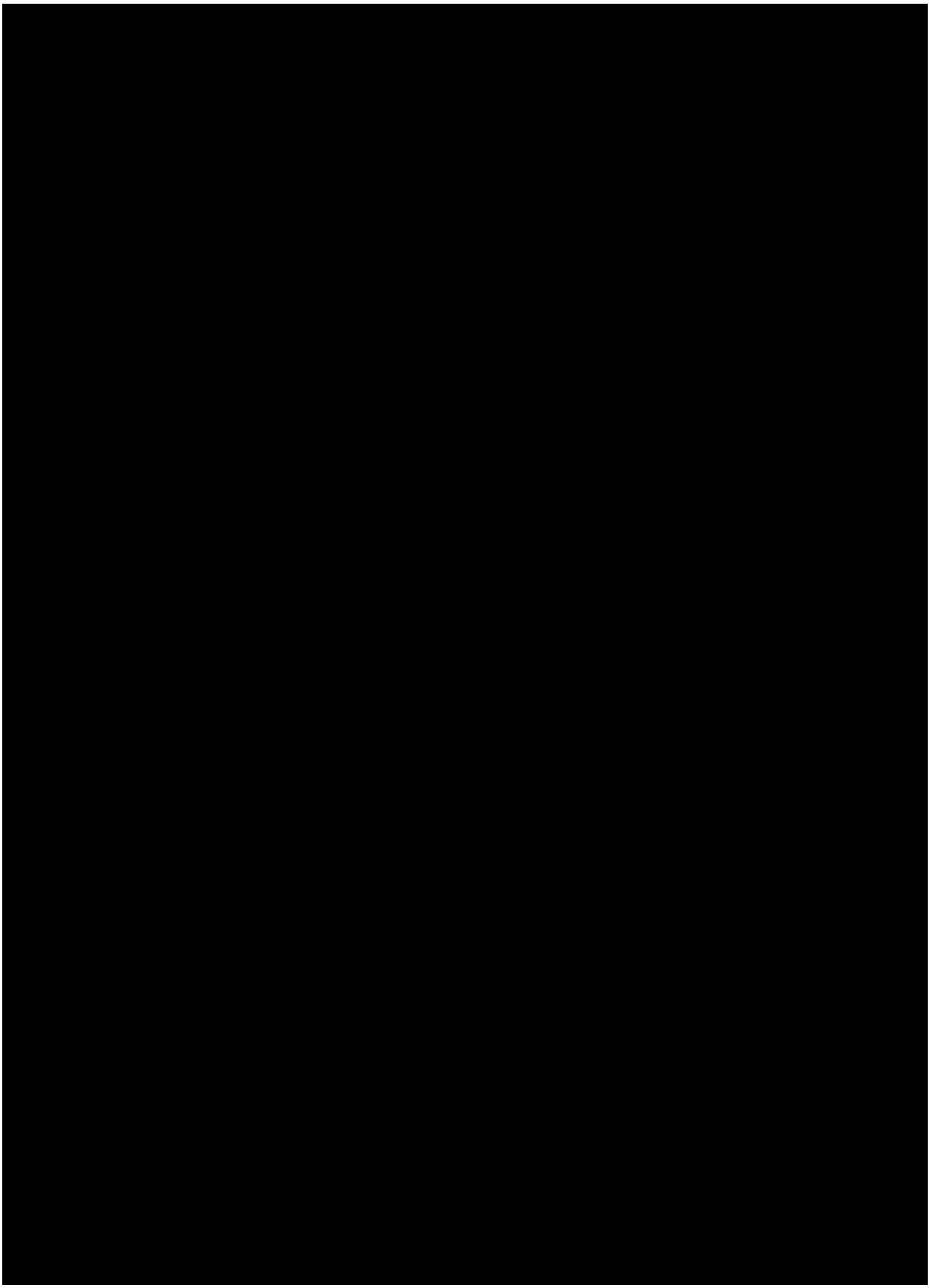
now = datetime.datetime.now()

print('Long day name :', now.strftime('%A'))
print('Short day name :', now.strftime('%a'))
print('Long month name :', now.strftime('%B'))

print('Short month name:', now.strftime('%b'))
print('Date time :', now.strftime('%c'))
print('Day of month :', now.strftime('%d'))

print('Hour number 24 :', now.strftime('%H')) print('Hour number 12 :', now.strftime('%I')) print('Day of year
print('Month of year
print('Minutes
print('AM or PM
print('Seconds
print('Short date
print('Short time

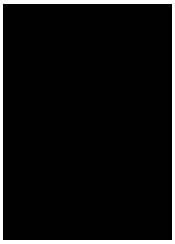
```
print( 'Short year
print( 'Long year
:', now.strftime('%j') ) :', now.strftime('%m') ) :', now.strftime('%M') )
:', now.strftime('%p') ) :', now.strftime('%S') ) :', now.strftime('%x') )
:', now.strftime('%X') ) :', now.strftime('%y') ) :', now.strftime('%Y') )

my_format = '%d/%m/%Y - %I:%M:%S %p'
:', now.strftime(my_format) ) print( 'Date time 12)
```



```
Long day name   : Monday
Short day name  : Mon
Long month name : July
Short month name: Jul
Date time       : Mon Jul 29 17:54:10 2019
Day of month    : 29
Hour number 24  : 17
Hour number 12  : 05
Day of year     : 210
Month of year   : 07
Minutes         : 54
AM or PM        : PM
Seconds         : 10
Short date      : 07/29/19
Short time      : 17:54:10
Short year      : 19
Long year       : 2019
Date time 12    : 29/07/2019 - 05:54:10 PM
>>>
```

Open an existing text file to read from to open an existing file

open utilization what follows has been done



The code

file = open('my_file.txt')

file.close()

Implementation

The file is opened

and closed, and if it does not exist an error occurs

Create a blank text file to write to It means that the file will be

created, use w What follows has been done

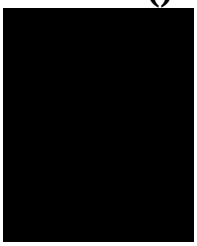
If it is, it will be deleted and then a new file created. Note that it

must after using it close the file with a function

The code

file = open('my_file.txt' , 'w')

file.close()



Then shut it down just in the same location as the code file

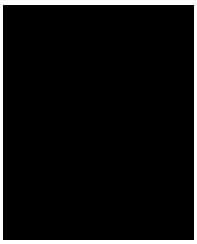
Implementation The file is created

Create a text file and write to it on the file using the write

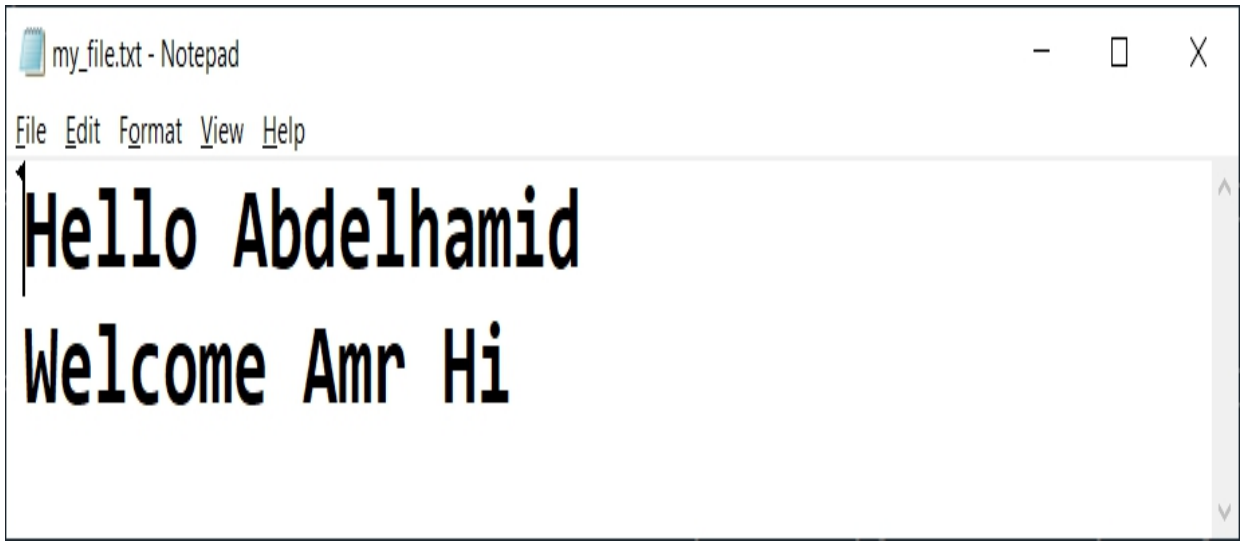
The following is a speech added

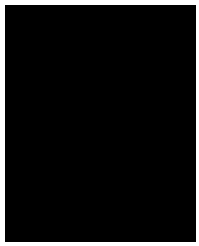
The code


```
file = open( 'my_file.txt' , 'w' )
file.write('Hello Abdelhamid\nWelcome Amr')
file.write(' Hi')
file.file.close()
```



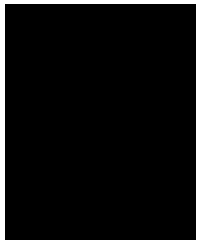
Implementation





The code


```
file = open( 'my_file.txt' , 'w' )
names = ['Amr','Ali','Ezz']
file.writelines(names)
file.close()
```



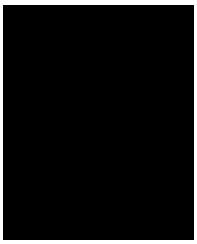
Regular reading from an existing file

The code


```
f = open( 'my_file.txt' , 'r' )
names = f.read()
```

f.close()
print(names)

Implementation


Read from an existing, single-line file to fetch the file content in

read lines use what follows has been done list

Code List


```
f = open( 'my_file.txt' , 'r' )
names = f.readlines()
f.close()
print( names )
```

Implementation

Read from an existing, multi-line file where each line has

content read lines use What follows has been done Any new line in each item With the addition of \ n list from the contents of the list

The code


```
f = open( 'my_file.txt' , 'r' )
names = f.readlines()
f.close()
print( names )
```

[redacted] Implementation


```
['Adel\n', 'Amr\n', 'Ali\n', 'Ezz']
>>>
```

[redacted]

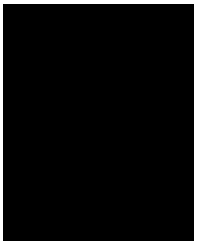
Reading from a file from a specific location to stop the reading indicator in place seek use What follows has been done you read

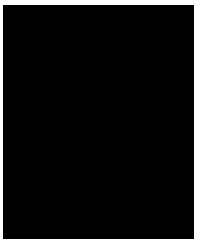
to the end read Main before reading, luck that

The code


```
f = open( 'my_file.txt' , 'r' )
f.seek(2)
names = f.read()
f.close()
print( names )
```

Implementation





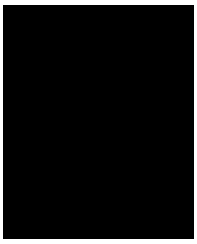
Read from a file line-by-line which begins with the first reading

read line use what follows has been done line, then next, then next, and so on

The code


```
f = open( 'my_file.txt' , 'r' )
print( f.readline() )
print( f.readline() )
print( f.readline() )
print( f.readline() )
f.close()
```

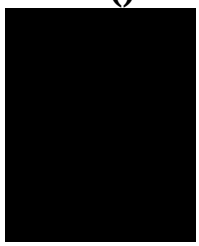

Implementation


Upon arrival false be zero or b readline function to the end of

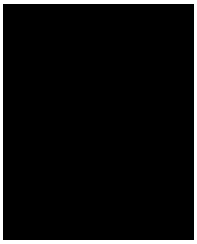
the file

The code


```
f = open( 'my_file.txt' , 'r' )
line = True
while line:

line = f.readline()
print( line )
f.close()
```


Implementation

```
Adel

Amr

Ali

Ezz

>>>
```



Continue writing to an existing text file writing supplement append to work use a what follows has been done on the file

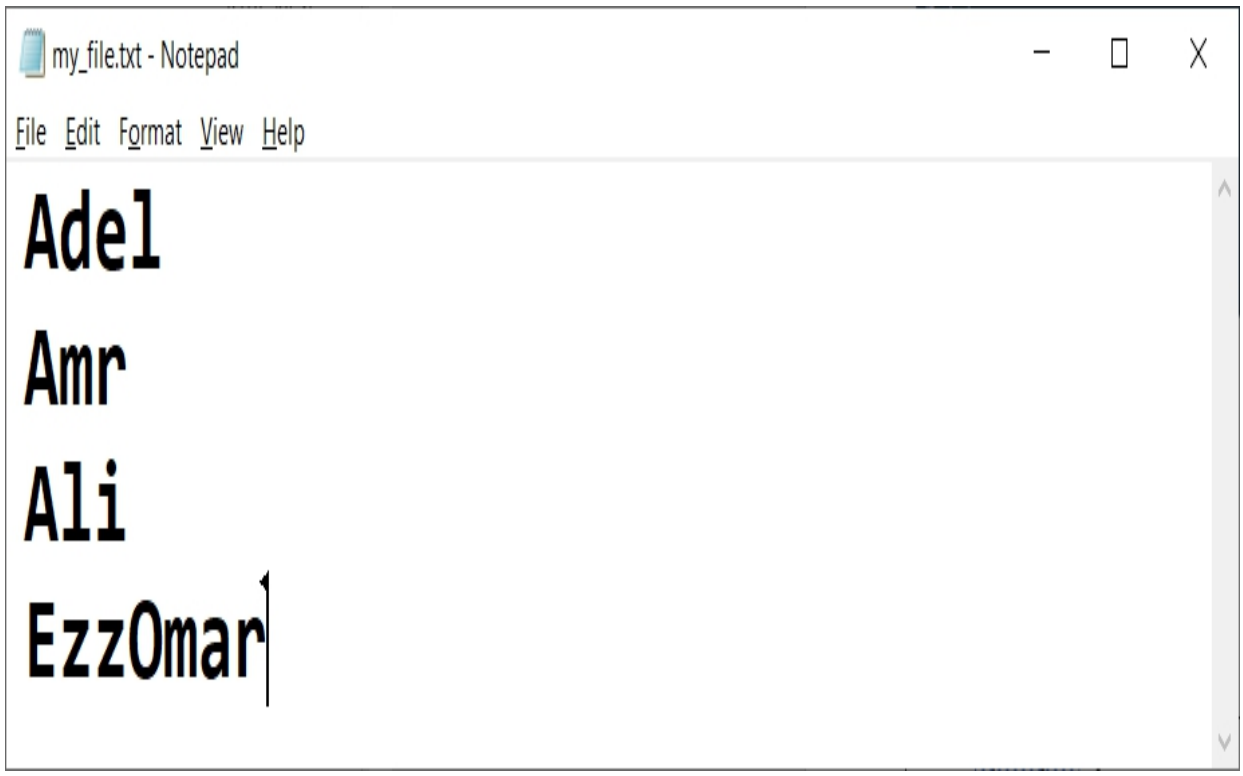
```
append to work use a what follows has been done on the file
```

without prejudice to the old speech

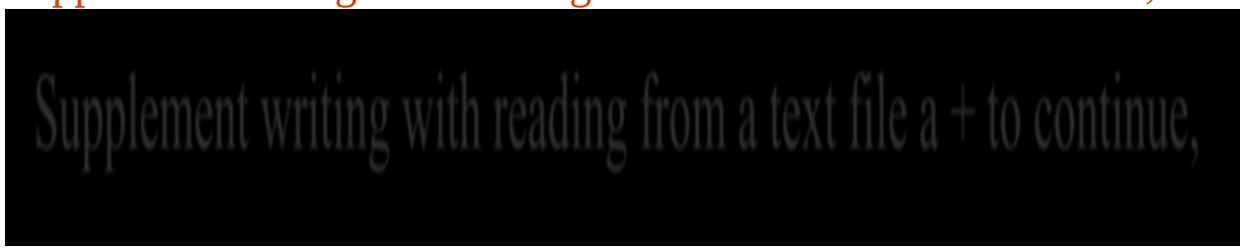
The code


```
f = open( 'my_file.txt' , 'a' )
f.write('Omar')
f.close()
```

Implementation



Supplement writing with reading from a text file a + to continue,

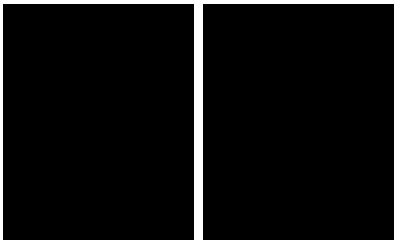


and with it can be read use what follows has been done

The code


```
f = open( 'my_file.txt' , 'a+' )
f.seek(0)
print( f.readlines() )
f.write('\nOmar')
f.seek(0)
print( f.readlines() )
f.close()
```

Implementation



Reading with writing to a text file to write and with it can read

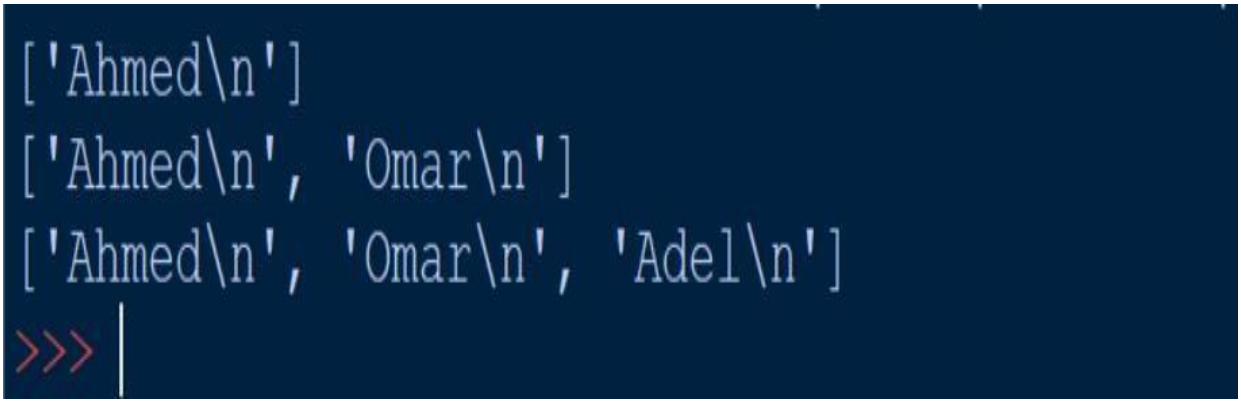
use w + what follows has been done

The code


```
f = open( 'my_file.txt' , 'w+' )
f.write('Ahmed\n')
f.seek(0); print( f.readlines() )
f.write('Omar\n')
```

f.seek(0); print(f.readlines())
f.write('Adel\n')
f.seek(0); print(f.readlines())
f.close()

Implementation



Writing with reading r + to read and with it can write

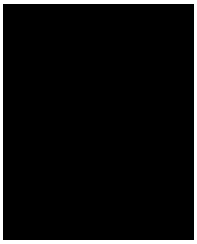
use what follows has been done

The code

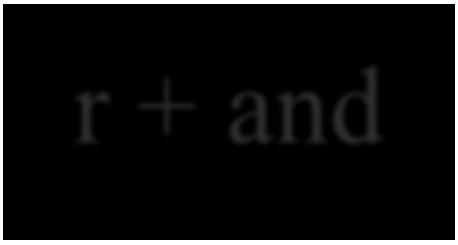

```
f = open( 'my_file.txt' , 'r+' )
f.seek(0); print( f.readlines() )
f.write('Ammar\n')
f.seek(0); print( f.readlines() )
f.write('Yaser\n')
f.seek(0); print( f.readlines() )
f.close()
```

Implementation

```
['Ahmed\n', 'Omar\n', 'Adel\n']
['Ahmed\n', 'Omar\n', 'Adel\n', 'Ammar\n']
['Ahmed\n', 'Omar\n', 'Adel\n', 'Ammar\n', 'Yaser\n']
>>>
```



Read, write and complete a binary file r + and r and w

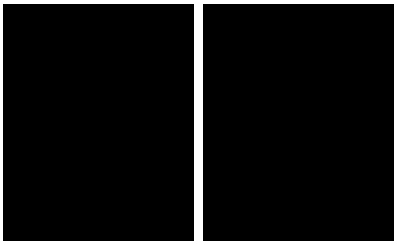


use the same previous patterns as what follows has been done

binary then b and others but with an extra character

The code


```
f = open( 'my_file.jpg' , 'wb' )
f = open( 'my_file.jpg' , 'rb' )
f = open( 'my_file.jpg' , 'wb+' )
f = open( 'my_file.jpg' , 'rb+' )
f = open( 'my_file.jpg' , 'ab' )
f = open( 'my_file.jpg' , 'ab+' )
f.close()
```



The code is here for knowledge, and then deal with it as per request for the binary file

Things after use to close with wholesale with the file is closed

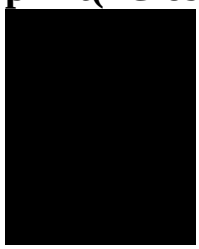
automatically because it was used with what follows has been done

The code

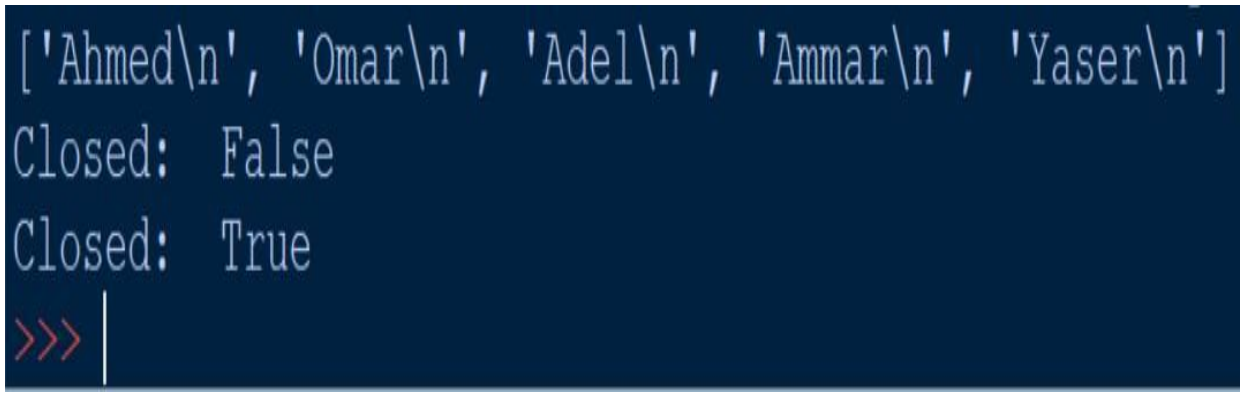

```
with open('my_file.txt','r') as file:
print( file.readlines() )
```

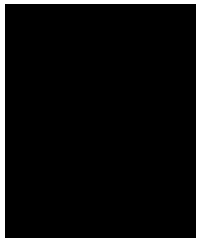
```
print( 'Closed: ', file.closed )
```

print('Closed: ', file.closed)



Implementation





Folders check for and create a file or folder

The code

import os

```
if not os.path.exists('My New Folder 1'):
os.mkdir('My New Folder 1')

if not os.path.exists('My New Folder 2'):
os.makedirs('My New Folder 2')
if not os.path.exists('my_file.txt'):
f = open( 'my_file.txt' , 'w' )
f.close()
```

They exist implementation the folders and file will be created if not

Delete files and folders

The code


```
import os
```

```
if os.path.exists('My New Folder 1'):
os.rmdir('My New Folder 1')
if os.path.exists('My New Folder 2'):
os.rmdir('My New Folder 2')
if os.path.exists('my_file.txt'):
os.remove('my_file.txt')
```

They exist if execute folders and file will be deleted

Located in a specific folder and folders find out about files It comes with all files and folders for the specified path list dir.

Function informs if a file isfile Have inside parentheses, and

notice that the function ML code


```
import os
files = os.listdir('New Folder')
for file in files:

if os.path.isfile('New Folder/' + file):
print('File :', file )
else:
print('Folder:', file )
```


```
File  : New Bitmap Image.bmp
Folder: New folder
Folder: New folder (2)
Folder: New folder (3)
File  : New Microsoft Excel Worksheet.xlsx
File  : New Text Document.txt
File  : New WinRAR archive.rar
>>>
```

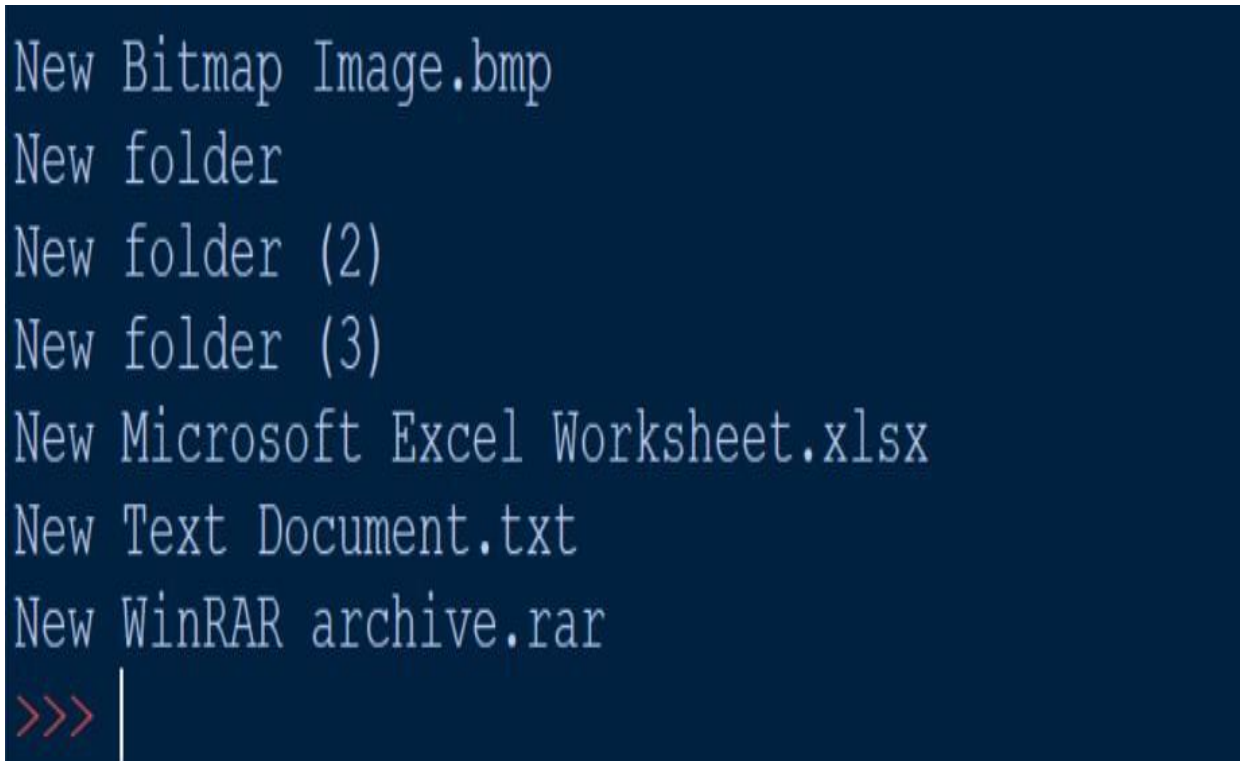
Create a list of files and folders

The code

import os
all_files = [f for f in os.listdir('New Folder')] for file in all_files:

print(file)

Implementation



The code

import os
os.system('mkdir my_folder_using_dos')

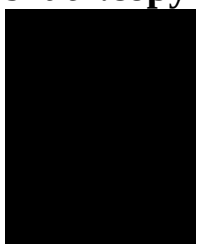
DOS, or any other as per Implementation A system folder will


be created Need.


Copying files

The code

import shutil
shutil.copy2('New Folder/Image.bmp' , 'NewFile.bmp')



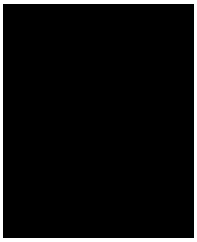
Implementation

The file is copied from the first location to the second location

Cut File Transfer

The code


```
import shutil
shutil.move( 'New Folder/Image.bmp' , 'NewFile.bmp' )
```



Implementation

The file will be moved from the first place to the second place

Folders copies

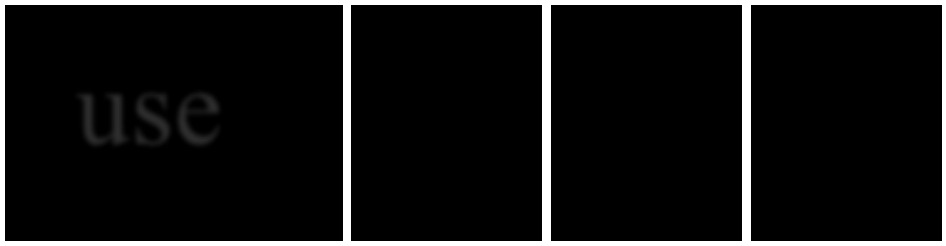
The code

import shutil
shutil.copytree('New Folder/New Folder (3)' , 'New')

Implementation

The volume is copied from the first location to the second location

Regular Expression use




```
import re
pattern = '^[A-Z][a-z]{1,15}$' text = input('Enter first name:') v =
re.match( pattern , text ) if v != None: print( 'Correct' ) else: print(
'Incorrect' )
```

Execution 1

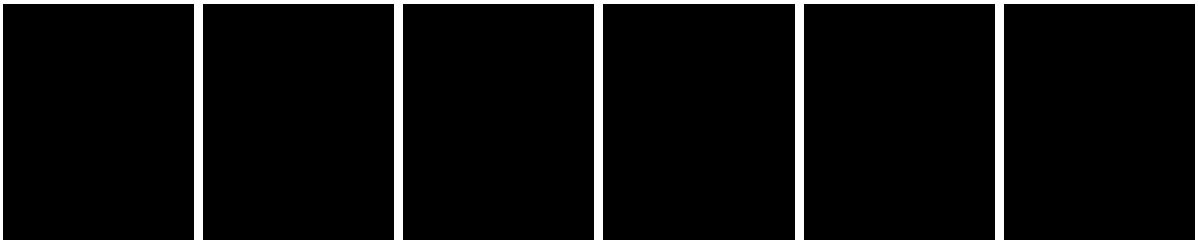
Execution 2


```
Enter first name:ahmed
Incorrect
>>>
```

Execution 3


```
Enter first name:Ahmed
Correct
>>>
```

Math use of mathematics



```
import math

print( 'PI print( 'SQRT print( 'Round print( 'Round print( 'Ceil print(
'Floor :', math.pi )
:', math.sqrt(81) ) :', round(1.5) )
:', round(1.4) )
:', math.ceil(1.1) ) :', math.floor(1.99) )

print( 'Absolute :', math.fabs(-50) )
print( 'Absolute :', abs(-10) )
print( 'Bower :', math.pow(5,3) )
print( 'Bower :', pow(5,3) )
print( 'Factorial:', math.factorial(5) )
print( 'Summation:', math.fsum( [1,2,3,4,5,6] ) ) print( 'Summation:',
sum(
[1,2,3,4,5,6] ) ) s =
[2500,6700,8400,2500,3400,210
0]
```

print('Average :', sum(s) / len(s))

Implementation

Implementation

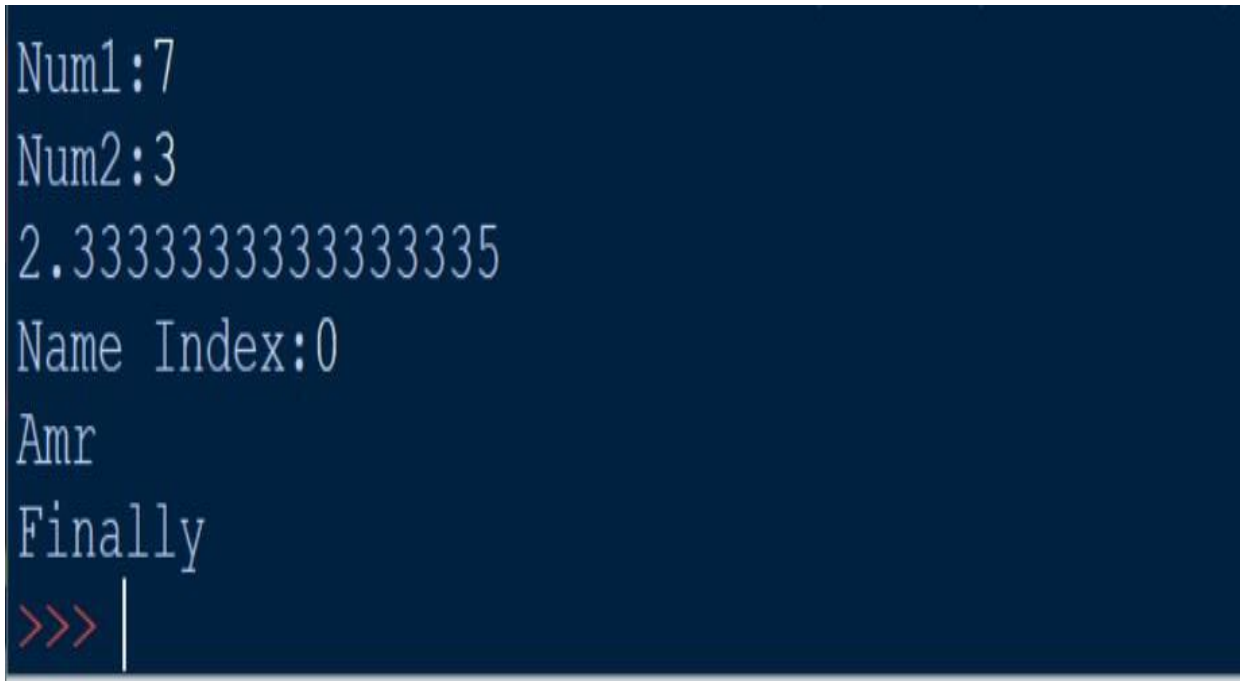
```
PI       : 3.141592653589793
SQRT     : 9.0
Round    : 2
Round    : 1
Ceil     : 2
Floor    : 1
Absolute : 50.0
Absolute : 10
Bower    : 125.0
Bower    : 125
Factorial: 120
Summation: 21.0
Summation: 21
Average  : 4266.666666666667
>>>
```

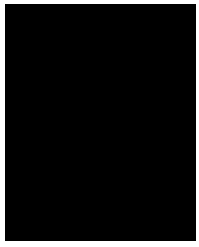
Error handling

The code


```
try:
num1, num2 = int(input('Num1:')) , int(input('Num2:')) result =
num1/num2
print( result )
names = ['Amr','Ali']
print( names[int(input('Name Index:'))] )

except ZeroDivisionError as ex1:
print( ex1 )
except IndexError as ex2:
print( ex2 )
except:
print( 'Unknown Error' )
finally:
print( 'Finally' )
```

Execution 1

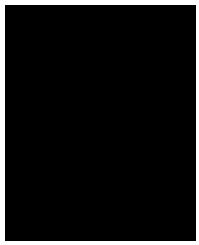




Execution2


```
Num1:7
Num2:0
division by zero
Finally
>>>
```

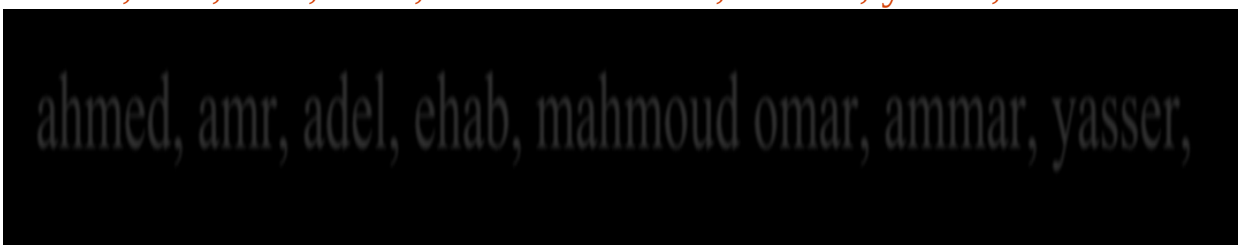
Execution 3



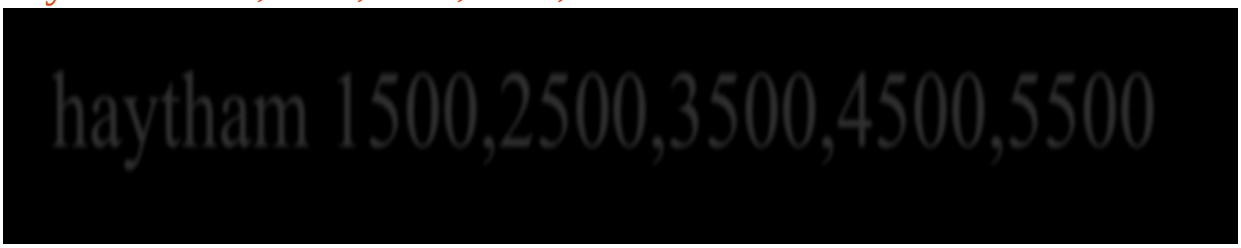
```
Num1:7
Num2:3
2.3333333333333335
Name Index:2
list index out of range
Finally
>>>
```

Work with csv files it has the following information

ahmed, amr, adel, ehab, mahmoud omar, ammar, yasser,



haytham 1500,2500,3500,4500,5500



info.csv create a file First

The code


```
import csv

f = open('info.csv') r = csv.reader(f) g1 = next(r)
print(g1)
g2 = next(r)
print(g2)
g3 = next(r)
print(g3)

f.close()
```

Implementation

Do not do anything normal functions create functions for lack of

implementation define three functions that do not have pass codes it was done

The code


```
def my_func1():
'''This is my function 1'''
def my_func2():
'''This is my function 1'''
def my_func3():
pass
```

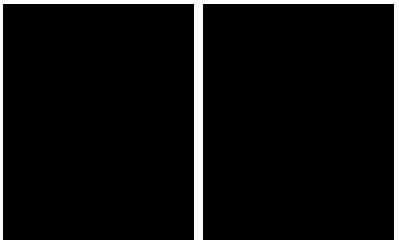
Implementation

Nothing will be done

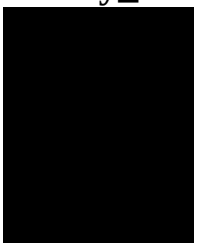
Create functions with codes and then call them

The code

def my_func1():



def my_func2():



'''This is my function 1'''

print('Welcome to function 2')

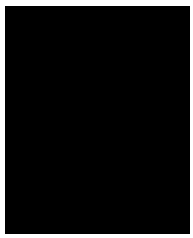
print('Function 2 is easy')

def my_func3():

 **print('Welcome to function 3')**  **print('Function 3 is easy')**

my_func2()

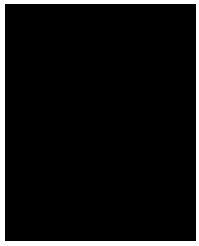
my_func3()



my_func1()

'''This is my function 1''' print('Welcome to function 1') print('Function 1 is easy')

```
Welcome to function 2
Function 2 is easy
Welcome to function 3
Function 3 is easy
Welcome to function 1
Function 1 is easy
>>>
```



Create a function that receives arguments But this function say_hello Make a function called What follows has been done

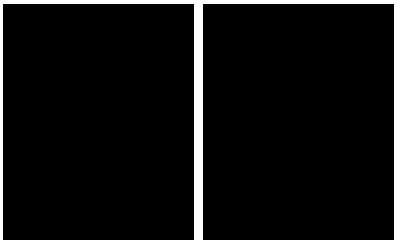
Then this variable is only used inside it name you receive a variable named Functions are useful because you use them more

than once and with different uses for the same function functions save time and effort, are very useful and should be

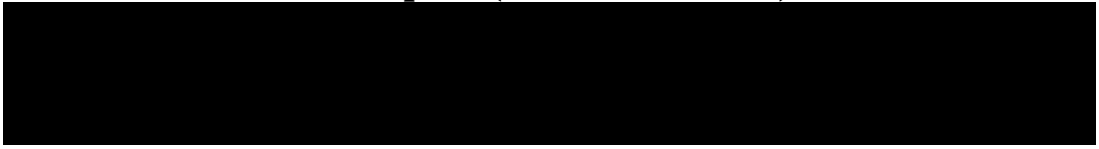
intensely focused that on her

The code

def say_hello(name):



print('Hello ' + name)



say_hello('Ahmed')

say_hello('Adel')

say_hello('Amr')

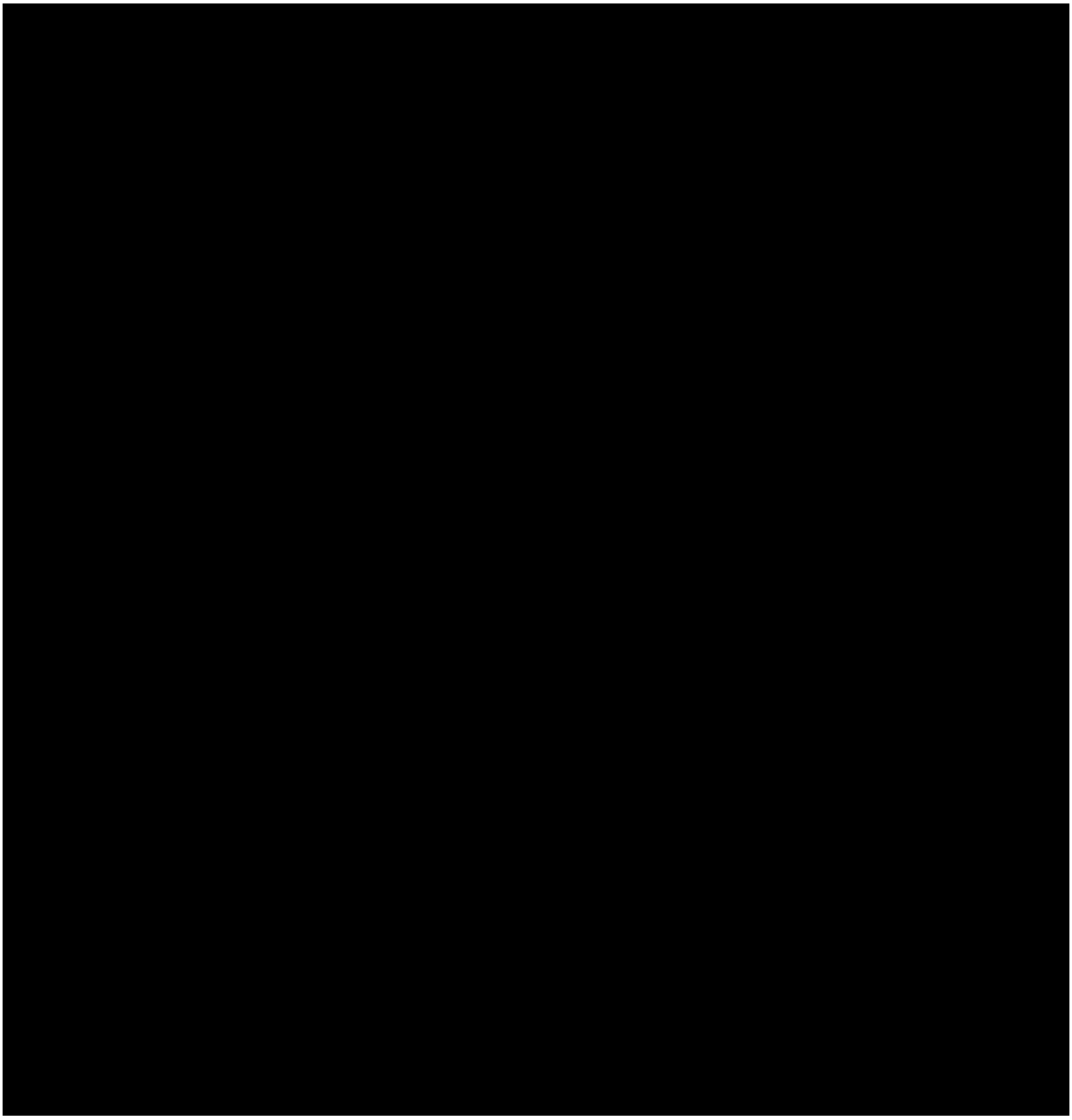
Create a function that receives the first number and the second

number to calculate what you want when called either What follows has been done operation and process It was addition,

subtraction, division, or multiplication in the same function

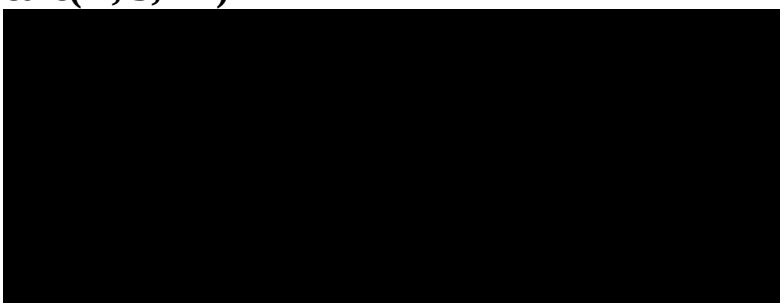
The code


```
def calc( num1 , num2 , ope ):
if ope == '+':
print( num1 + num2 )
elif ope == '-':
print( num1 - num2 )
elif ope == '*':
print( num1 * num2 )
elif ope == '/':
if num2==0: num2=1
```

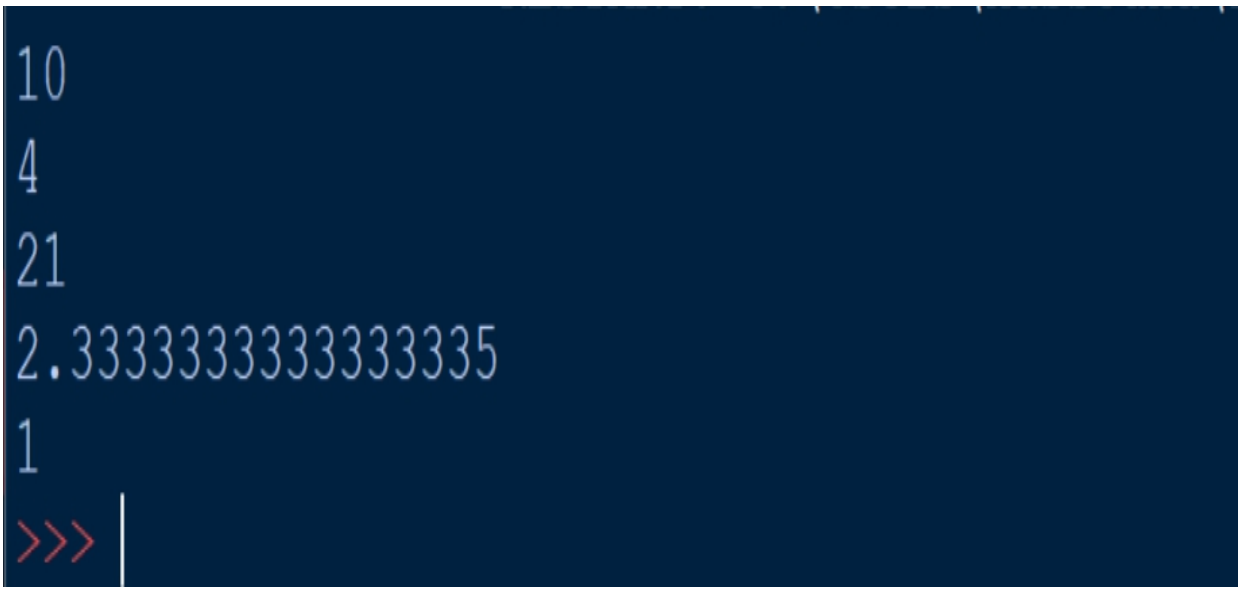


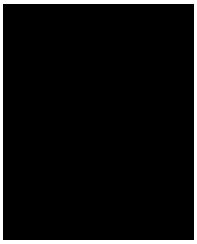
print(num1 / num2)
calc(7, 3, '+')

calc(7, 3, '-')



calc(7, 3, '*') calc(7, 3, '/') calc(7, 3, '%')





Create a function with infinite arguments use the function with

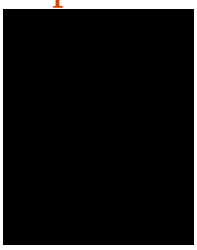
one or more arguments, and note that each What follows has

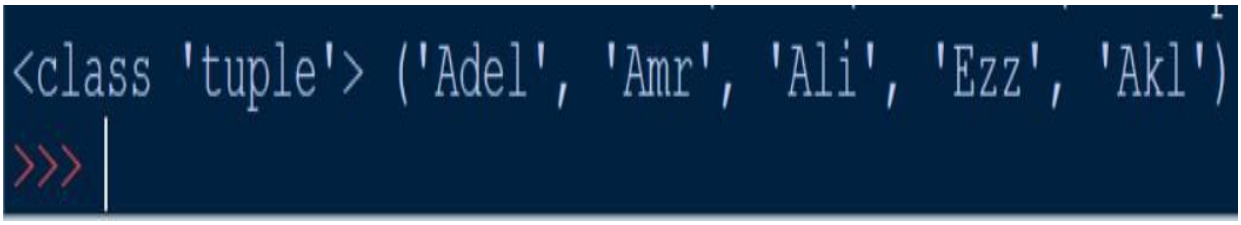
been done one tuple the media collects

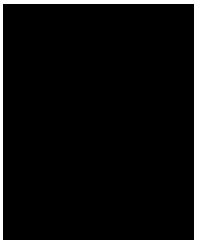
The code


```
def names(*names):
print( type(names), names )
names('Adel','Amr','Ali','Ezz','Akl')
```

Implementation






The code

def get_values(*values):
print(values)
for val in values:

print(type(val), val)

get_values(111,'Amr',4655.50,True)

[redacted]

[redacted]

```
(111, 'Amr', 4655.5, True)
<class 'int'> 111
<class 'str'> Amr
<class 'float'> 4655.5
<class 'bool'> True
>>>
```

[redacted] Create a function that returns a value returns a value, that is,

when calling this function what follows has been done

It will return a value that can be printed or put into a variable it

has no value return the code after return

The code

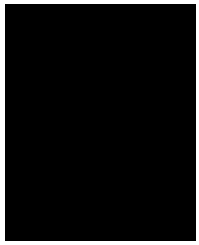

```
def my_func():
print('Before Return')
return 'Test Return'
print('After Return')

my_return_val = my_func()
print( my_return_val )
```

Implementation

def say_hello(name): return 'Hello ' + name

print(say_hello('Ahmed')) print(say_hello('Adel')) print(say_hello('Amr'))

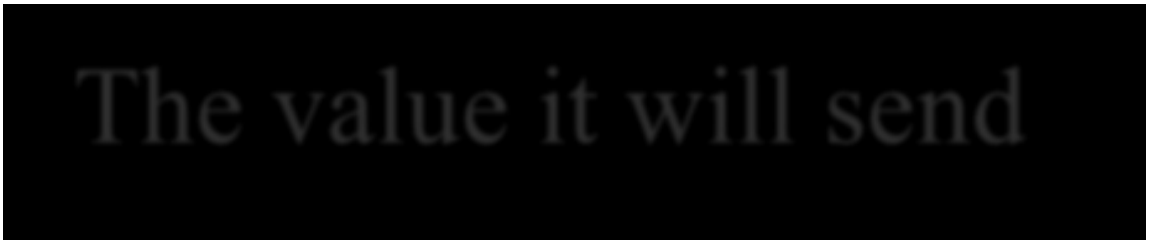
Implementation



Create a function based on the idea of the dictionary here it is used to return a value for a dictionary inside the function, such

that A function that is to be returned from the dictionary

The value it will send



The code


```
def num_name(number):
return {
0:'zero', 1:'one', 2:'two', 3:'three',

4:'four', 5:'five', 6:'siz', 7:'seven',
8:'eight', 9:'nine', 10:'ten'
}[number]
```

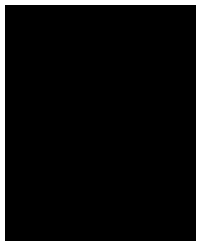
98

print(num_name(0))

print(num_name(1)) print(num_name(2)) print(num_name(3))

print(num_name(4)) print(num_name(5)) print(num_name(6)) print(num_name(7)) print(num_name(8)) print(num_name(9)) print(num_name(10))

```
zero
one
two
three
four
five
siz
seven
eight
nine
ten
>>>
```

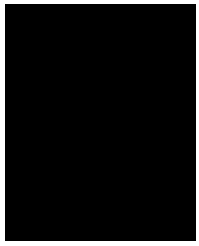


As if it were a function Lambda use expressions

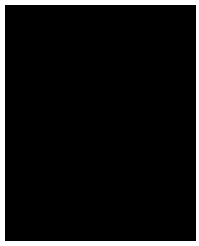
As if it were a function Lambda use expressions

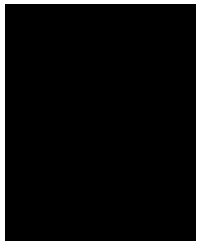
The code


```
calc = lambda num1, num2: num1 + num2
print( calc(7,3) )
print( calc(5,8) )
print( calc(6,9) )
```



Implementation





Create a reference function that calls itself create a function to

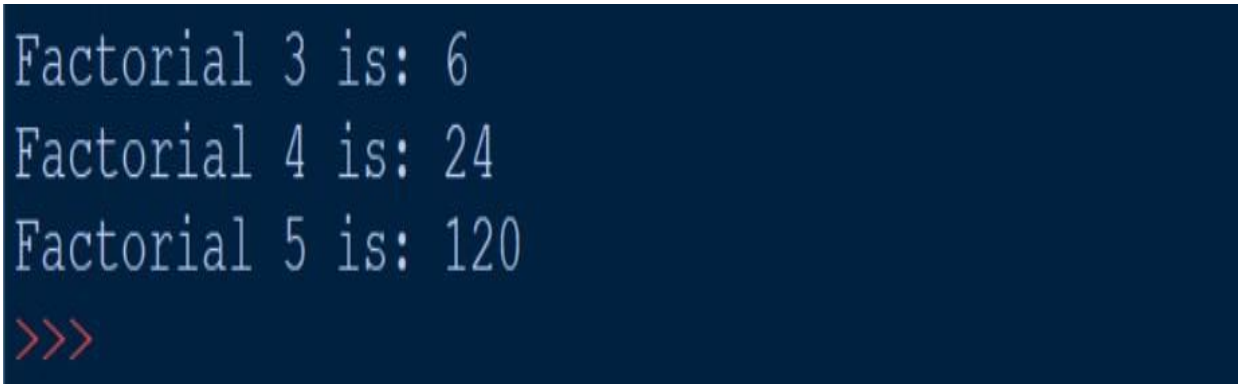
calculate and return the factorial what follows has been done

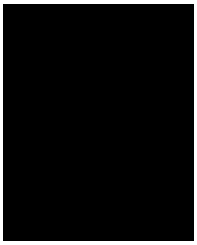
Factorial that the reference function has a special understanding,

to understand it well we advise

```
def fact(num):
if num == 0: return 1
else: return num * fact( num - 1 )

print( 'Factorial 3 is:', fact(3) ) print( 'Factorial 4 is:', fact(4) ) print( 'Factorial 5 is:', fact(5) )
```

Implementation



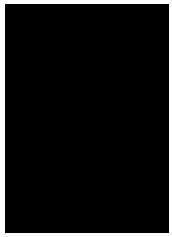


As an executable code library module build up in the same

location as the executable file that we are trying my.py first,

create a file named he who expresses the model

create a file named he who expresses the model and save it



my.py secondly, the basic codes have to be written for the file,

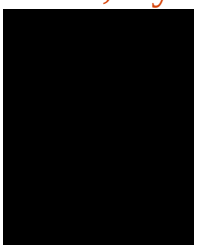
my.py secondly, the basic codes have to be written for the file,

and the following code should be written in the file

and the following code should be written in the file

```
def say_hello( name ):
print( 'Hello ' + name )
def calc( num1 , num2 , ope ): if ope == '+':
print( num1 + num2 ) elif ope == '-':
print( num1 - num2 ) elif ope == '*':
print( num1 * num2 ) elif ope == '/':
if num2==0: num2=1 print( num1 / num2 ) elif ope == '%':
if num2==0: num2=1 print( num1 % num2 )
```

Third, try the following on the main code file



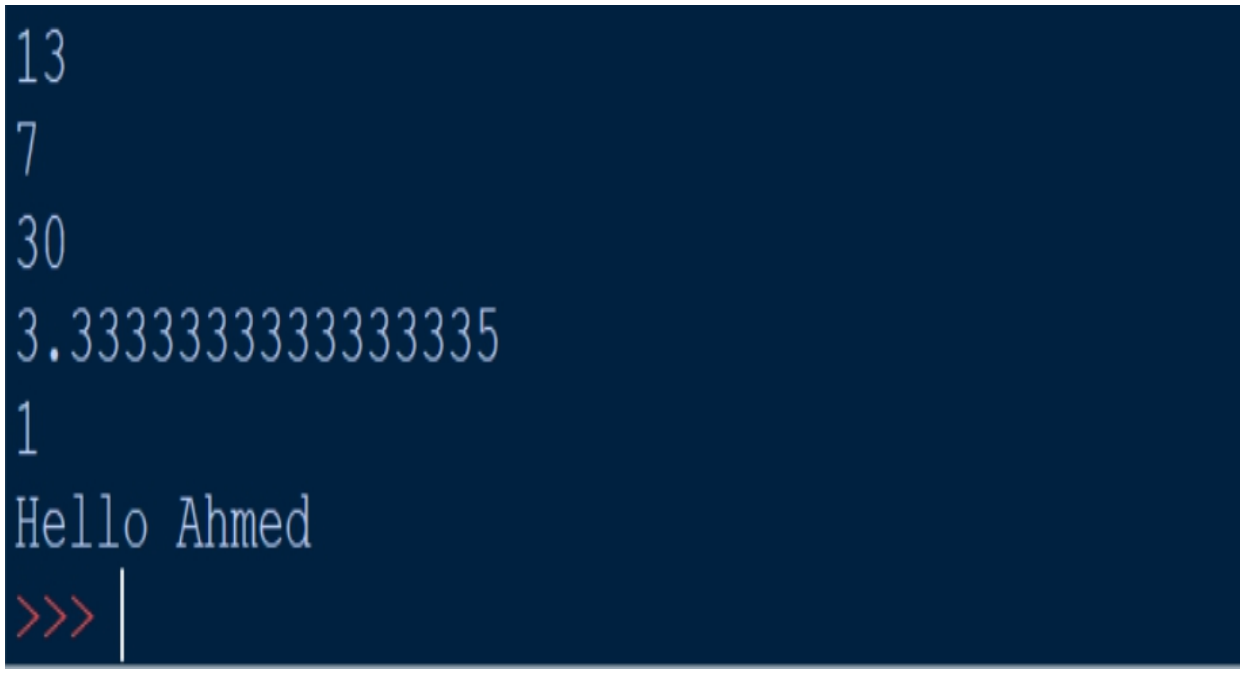
The code


```
import my
my.calc(10,3,'+')
my.calc(10,3,'-')
my.calc(10,3,'*')
my.calc(10,3,'/')
my.calc(10,3,'%')
my.say_hello('Ahmed')
```

[redacted] Implementation

[redacted] Implementation [redacted]

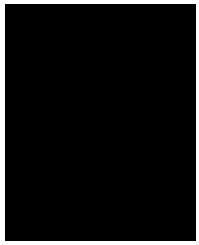
[redacted] [redacted]



[redacted] [redacted] The code

```
from my import calc
from my import say_hello
calc(81,18,'+')
calc(150,51,'-')
calc(33,3,'*')
calc(495,5,'/')
calc(23,12,'%')
say_hello('Adel')
```

Implementation



Execute codes from an external file or from text in the same

place as the executable mycode.py first, create a file named

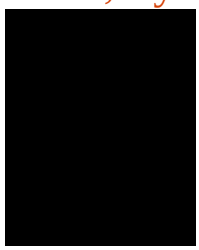
on which we try the basic codes and save it mycode.py

second, write the following code in the file

print('Test From mycode.py')
for x in range(3):
print('Test From mycode.py ' + str(x)) def calc(num1 , num2 , ope):
if ope == '+':
print(num1 + num2)

```
elif ope == '-':
print( num1 - num2 )
elif ope == '*':
print( num1 * num2 )
elif ope == '/':
if num2==0: num2=1
print( num1 / num2 )
elif ope == '%':
if num2==0: num2=1
print( num1 % num2 )
```

Third, try the following on the main code file



The code


```
exec( open('mycode.py').read() )
exec( 'print("This Code from string")' )
```

Implementation

```
Test From mycode.py
Test From mycode.py 0
Test From mycode.py 1
Test From mycode.py 2
This Code from string
>>>
```

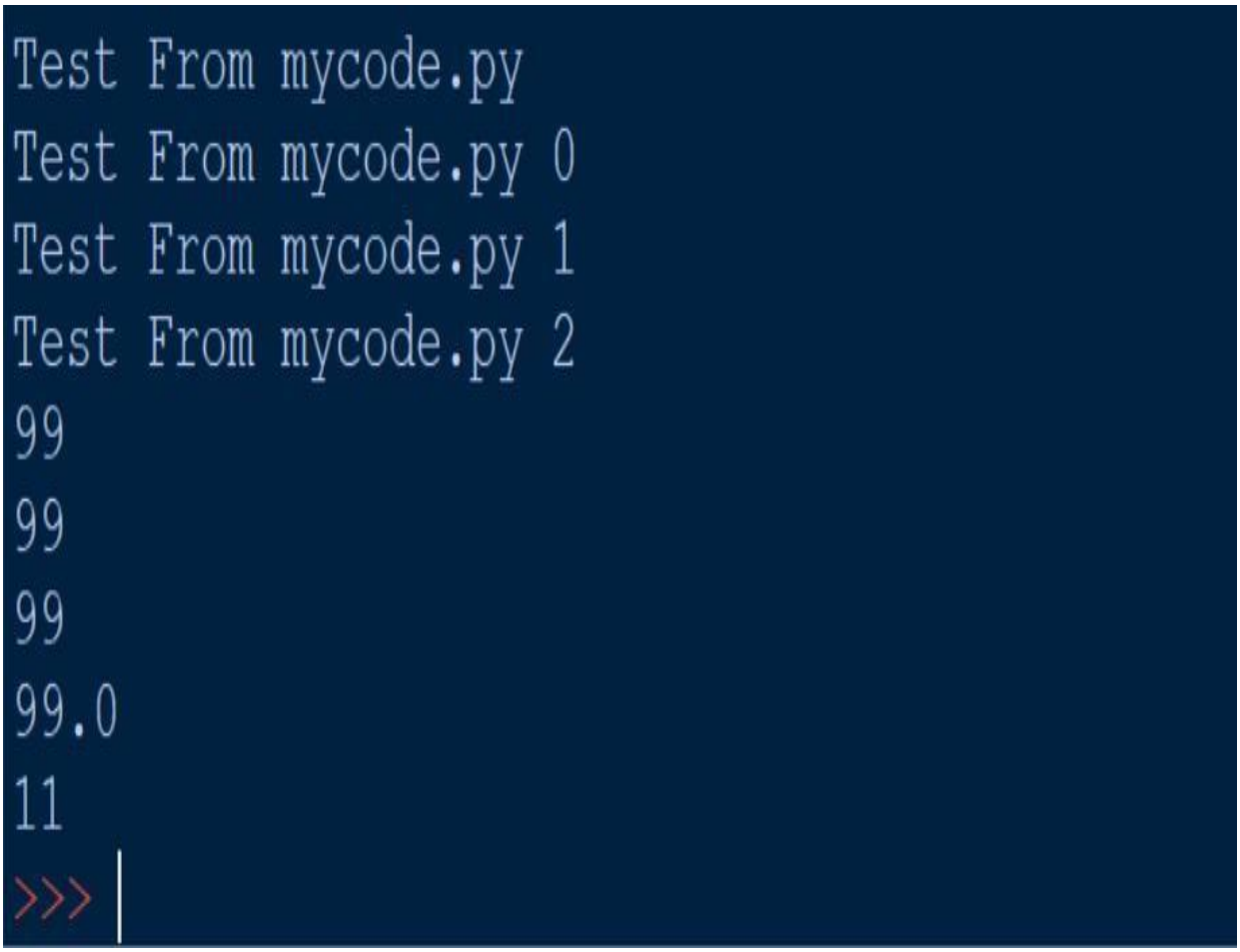

The code

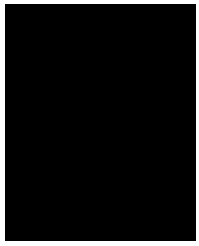

exec(open('mycode.py').read())


calc(81,18,'+')

calc(150,51,'-')
calc(33,3,'*')

calc(495,5,'/') 103
calc(23,12,'%')

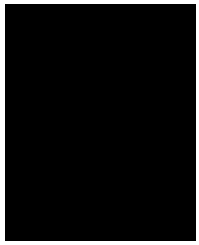




The code


```
exec("""
for x in range(3):
print( 'X=' + str(x) )
""")
```

Implementation



Show help to show help to me help use the function what follows has been done A topic, whether you are of a craft like

functions and others, or it exists such as print, str, tuple, etc., even if a module you made yourself or a module already exists

in the language, even if it is a variable, it tells you that

Uses equivalent in language

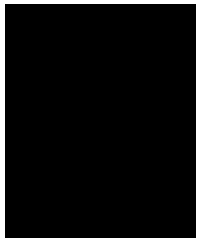
The code

def say_hello(name):
'''Send any name for say hello''' print('Hello ' + str(name))

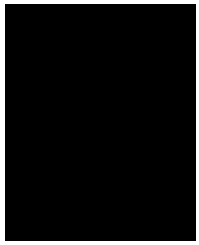
help(say_hello)

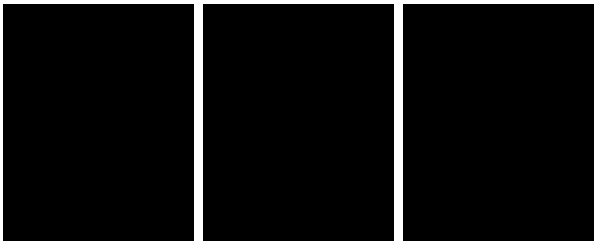
Implementation

```
Help on function say_hello in module __main__:

say_hello(name)
    Send any name for say hello

>>>
```


The code

help(print)


Implementation



```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

The code

import my

help(my)

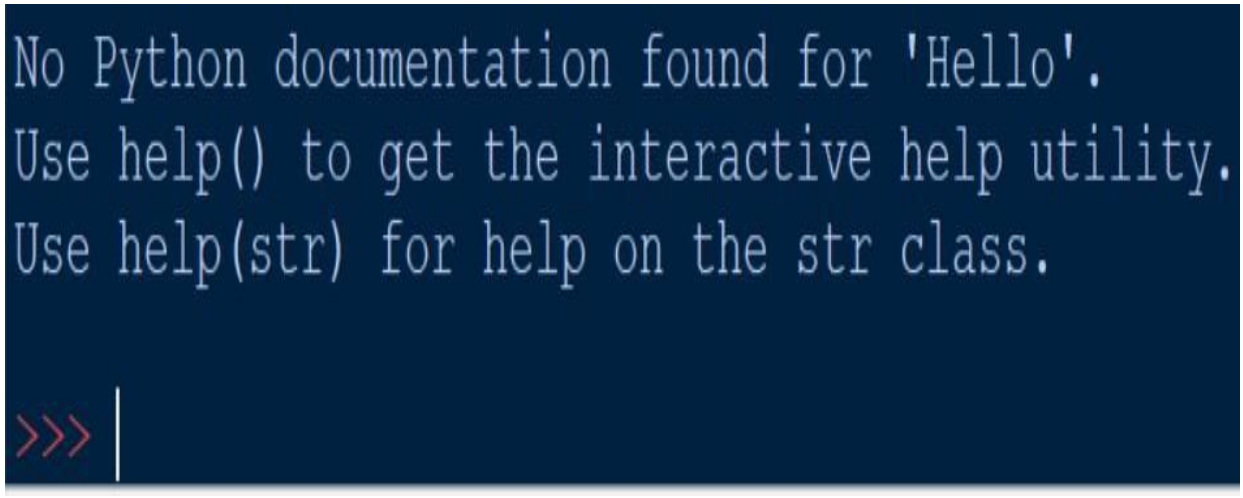
Implementation

```
Help on module my:

NAME
    my

FUNCTIONS
    calc(num1, num2, ope)

    say_hello(name)

FILE
    c:\users\usernowa\desktop\my.py


>>>
```

The code

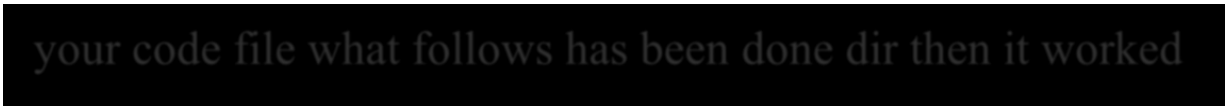
help('Hello')

Implementation



Exploring what is in the Python language dir show all users in

your code file what follows has been done dir then it worked

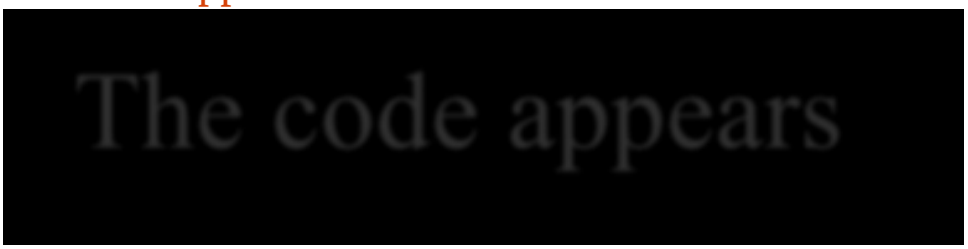


import or work when increasing a variable or function

it's about a specific variable or function dir with the existing and

can be searched complete that

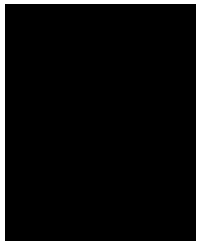
The code appears



print(dir())

Implementation

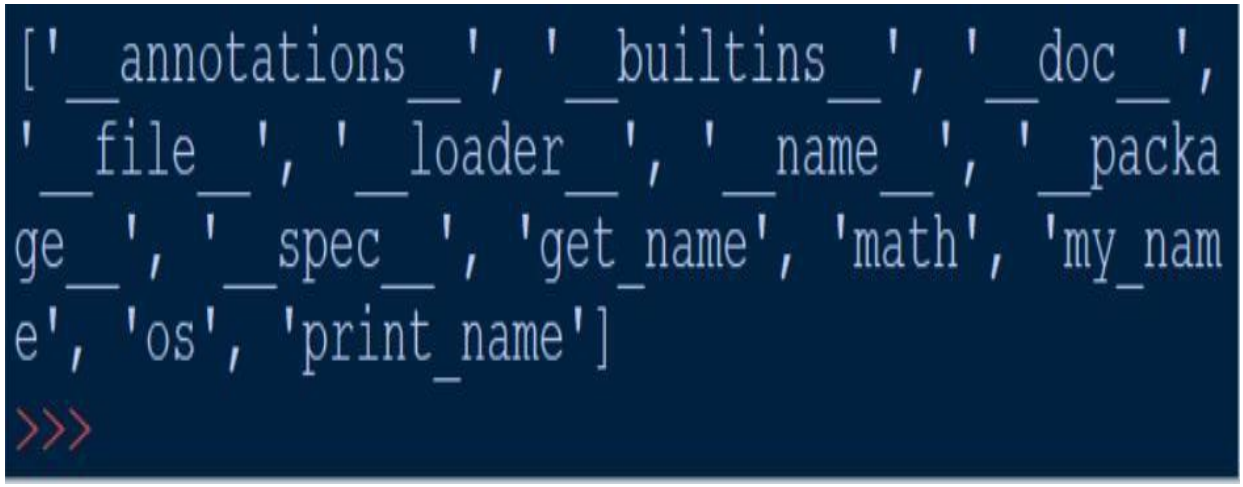

```
['__annotations__', '__builtins__', '__doc__',
'__file__', '__loader__', '__name__', '__packa
ge__', '__spec__']
>>>
```



The code


```
import math
import os
my_name = 'Mohamed'
def get_name():return 'Mohamed'
def print_name():print(get_name())
print( dir() )
```

[redacted] Implementation



[redacted] The code

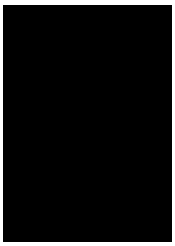

```
my_name = 'Mohamed'
def get_name():return 'Mohamed'
def print_name():print(get_name())
print( dir(get_name) )
```

Features only class build up and bears the number employee

callus worked in the name what follows has been done and

salary address and address name and name number active and

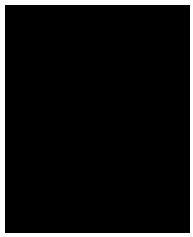
potency salary The properties inside the callus are variables



The code

class employee:
number = 0
name = ''

address = ''
salary = 0.0
active = True



Implementation

Nothing is running because the callus has not been used

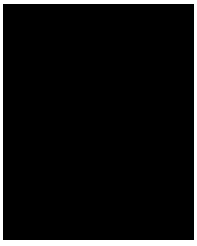
It has only class functions. Create functions for the class which

are functions, and start from receiving a variable for the lucky object!

That the code

class my_print:
def print1(self):
print('Test Print 1')

def print2(self):
print('Test Print 2')
def print3(self):
print('Test Print 3')



Implementation nothing is executed because it does not use a class

it has both features and functions a variable must be defined within each function in callus to express this class It follows

build up complete the variable about the object that will be defined, knowing that this variable will not be sent when calling

the function, but it is a default of the object call the elements of

a callus inside the functions using what follows has been done he is the famous or something else self the default variable for

the function, either we call it

The code


```
class employee:
number = 0
name = ''
address = ''
salary = 0.0
active = True
def get_data(o):

info = str(o.number)+';'+o.name+';'+o.address info +=
';'+str(o.salary)+';'+str(o.active) return info

def print_data(o):
print( o.get_data() )
```

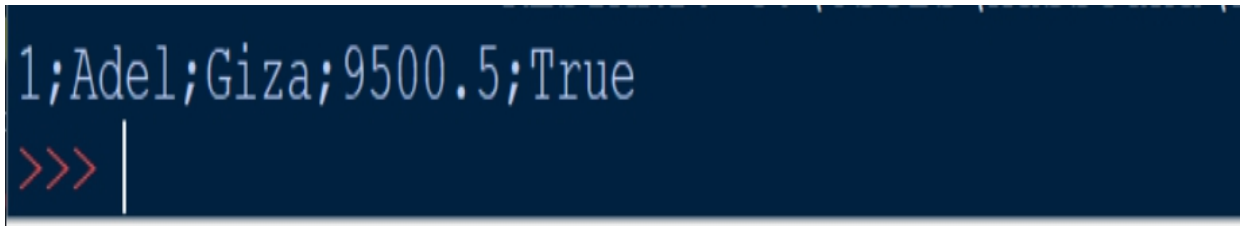
Nothing is running because the class has not been used

Use it class build up define an object Emp 1 From him as

The code


```
class employee:
number = 0
name = ''
address = ''
salary = 0.0
active = True
def get_data(o):

info = str(o.number)+';'+o.name+';'+o.address info +=
';'+str(o.salary)+';'+str(o.active) return info

def print_data(o):
print( o.get_data() )
emp1 = employee()
emp1.number = 1
emp1.name = 'Adel'
emp1.address = 'Giza'
emp1.salary = 9500.5
emp1.print_data()
```

Implementation



His object and more work class build up of the function color and model send two variables what follows has been done does

not exist self as if the variable set_data

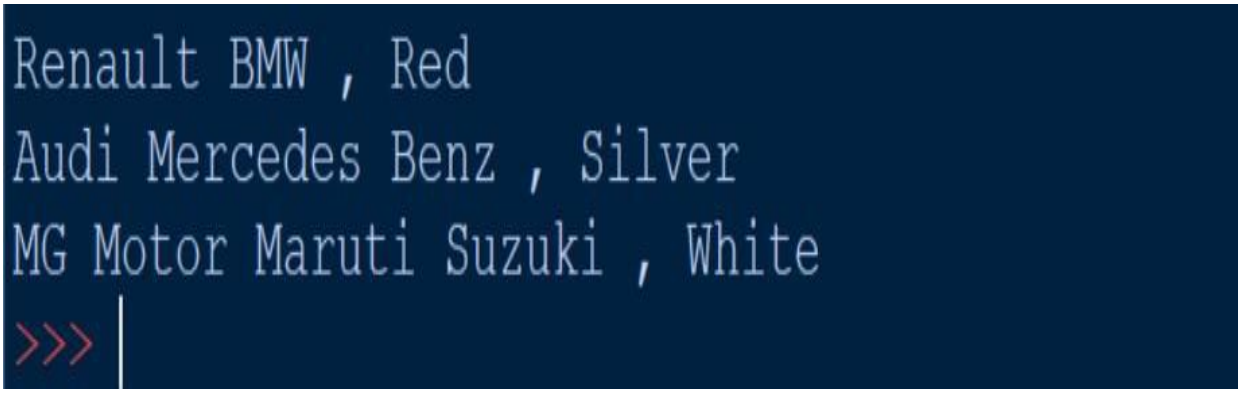
class car:
model = ''
color = ''
def set_data(self, model, color):

self.model = model
self.color = color
def get_data(self):

return self.model+' , '+self.color def print_data(self):
print(self.get_data())
car1 = car()
car2 = car()
car3 = car()
car1.set_data('Renault BMW','Red')
car2.set_data('Audi Mercedes Benz','Silver') car3.set_data('MG Motor Maruti Suzuki','White')

car1.print_data() car2.print_data() car3.print_data()

Implementation



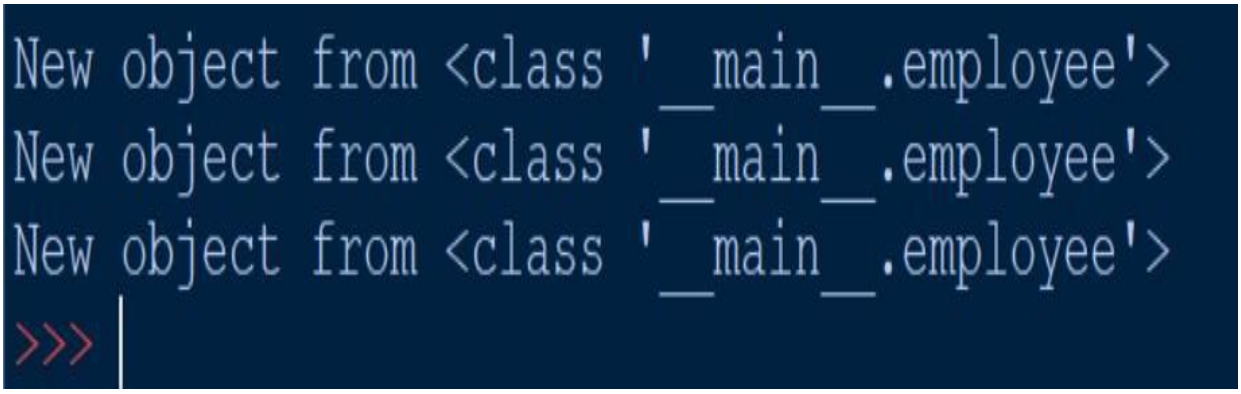
Building function in a class Write the init code for the class

what follows has been done this function is executed automatically when defining a new object employee

The code

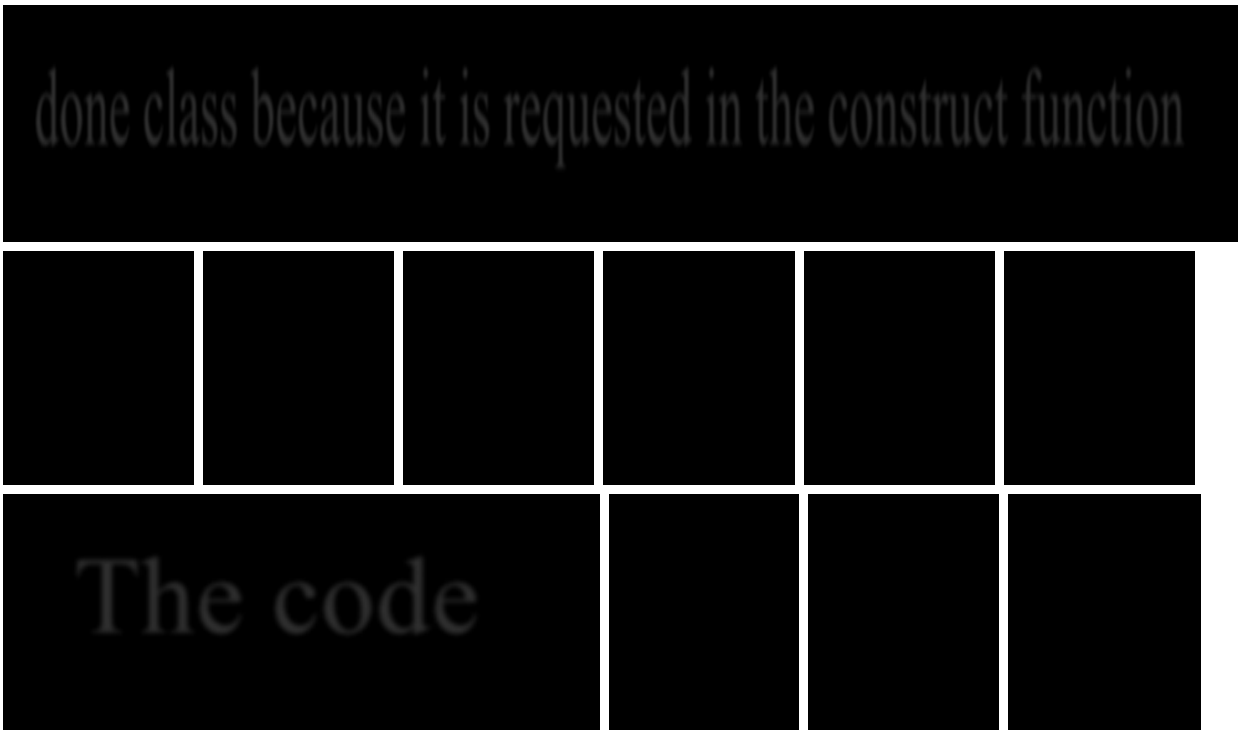

```
class employee:
def init (self):
print( 'New object from ' + str( type(self) ) )

emp1 = employee()
emp2 = employee()
emp3 = employee()
```

Implementation

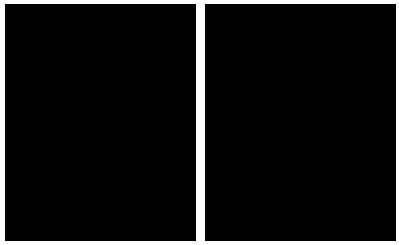


Arguments with a construct function media submission is mandatory, with each object defined from what follows has been

done class because it is requested in the construct function



Implementation

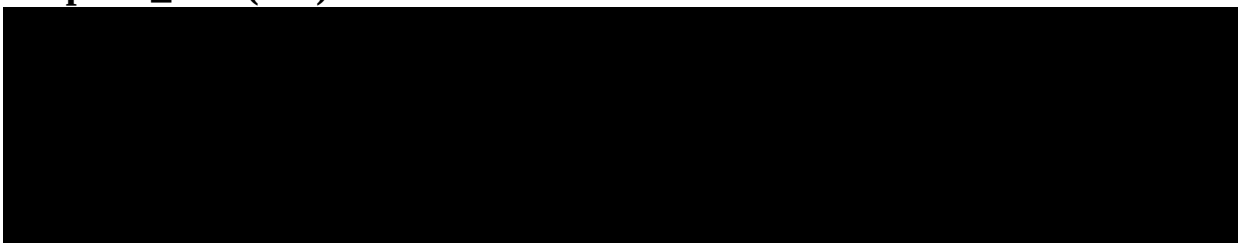


class employee:
emp_id = 0
emp_name = ''

def init (self, emp_id, emp_name):

self.emp_id = emp_id
self.emp_name = emp_name

def print_data(self):



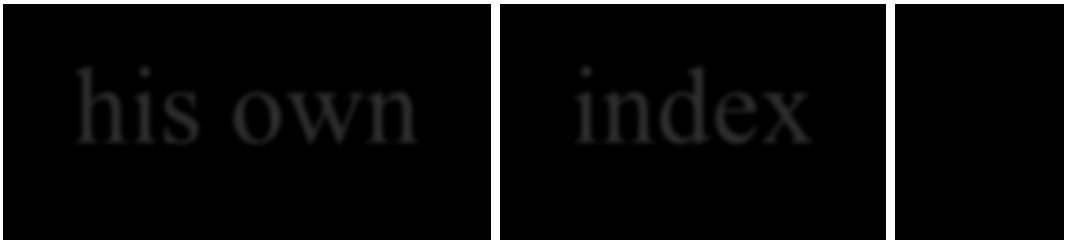
print(self.emp_id, self.emp_name)
emp1 = employee(1,'Ahmed') emp2 = employee(2,'Adel') emp3 = employee(3,'Amr')

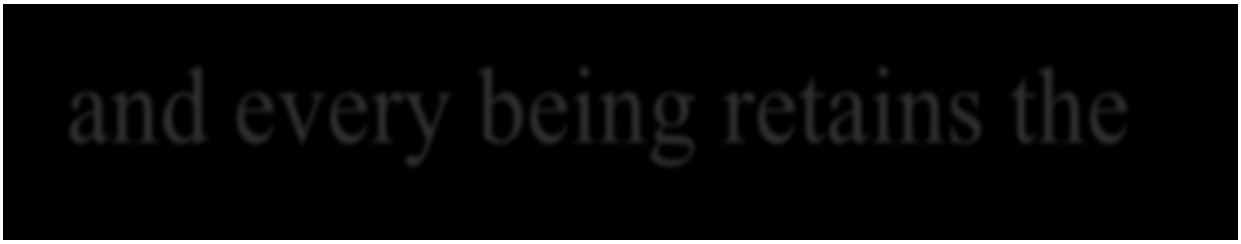
emp1.print_data() emp2.print_data() emp3.print_data()

Objects for the create a counter and index objects_count make a

counter for the number of objects what follows has been done

his own index and every being retains the



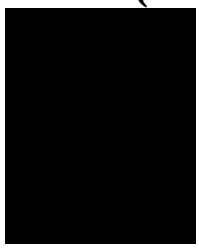


The code

from itertools import count

class employee:
objects_count = count(0)
index = 0

def init (self):



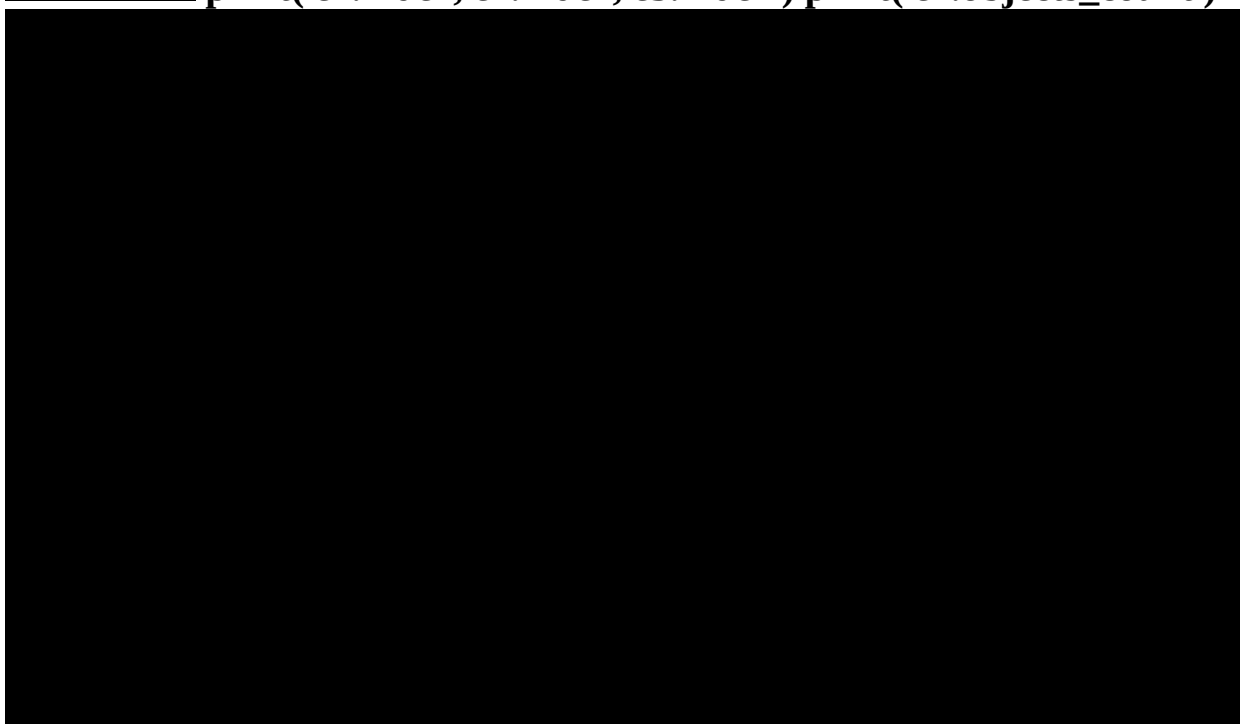
self.index = next(self.objects_count)

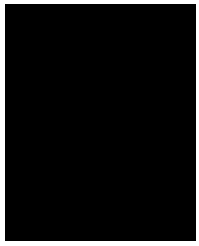
e1 = employee()

e2 = employee()

e3 = employee()

print(e1.index, e2.index, e3.index) print(e1.objects_count)





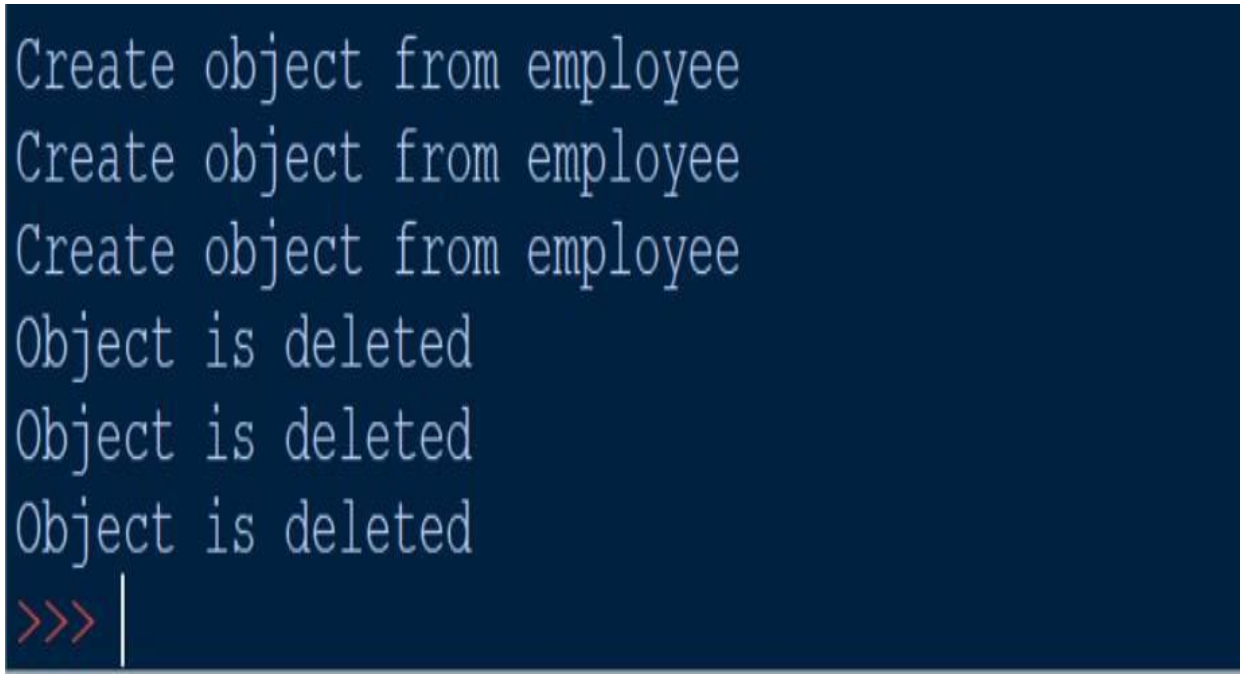
Implementation

The catabolism function in the class use the del demolition function to execute at what follows has been done the object delete

class employee:
def init (self):
print('Create object from employee')

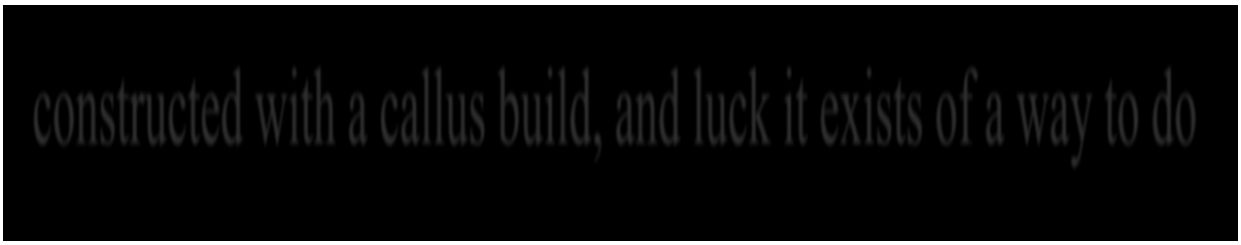
```
def del (self):
print('Object is deleted')
e1 = employee()
e2 = employee()
e3 = employee()
del e1
del e2
del e3
```

Implementation

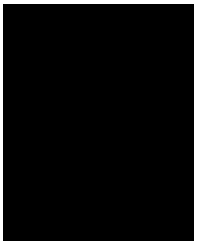


Include new properties for the object and the property number use the what follows has been done name although they are not

constructed with a callus build, and luck it exists of a way to do that more html code



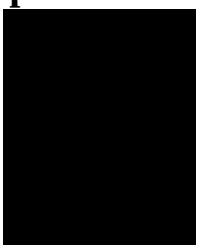
class employee:
pass
e1 = employee()
e1.number = 1
e1.name = 'Amr'
print(e1.number, e1.name)



The code

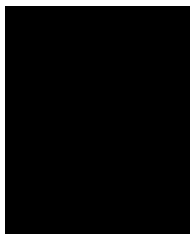
class employee:

pass



e1 = employee()

setattr(e1, 'number', 1)



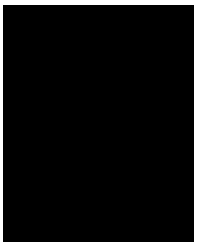
setattr(e1, 'name', 'Amr')
print(e1.number, e1.name)

Implementation

Include properties that implement code to execute a specific code appeal to the property only what follows has been done its like a function

The code


```
class employee:
pass
e1 = employee()
setattr( e1, 'test', exec('print("test")') )
e1.test
```



Create secret elements inside the class twice or underscore

everything begins with a symbol that more without adding an underscore in the end, it becomes a secret element, whether it is

a property or a function, and if it is class to use it, an error will occur

The code

class my_class:
my_attr1 = 'attr1'
my_attr2 = 'attr2'

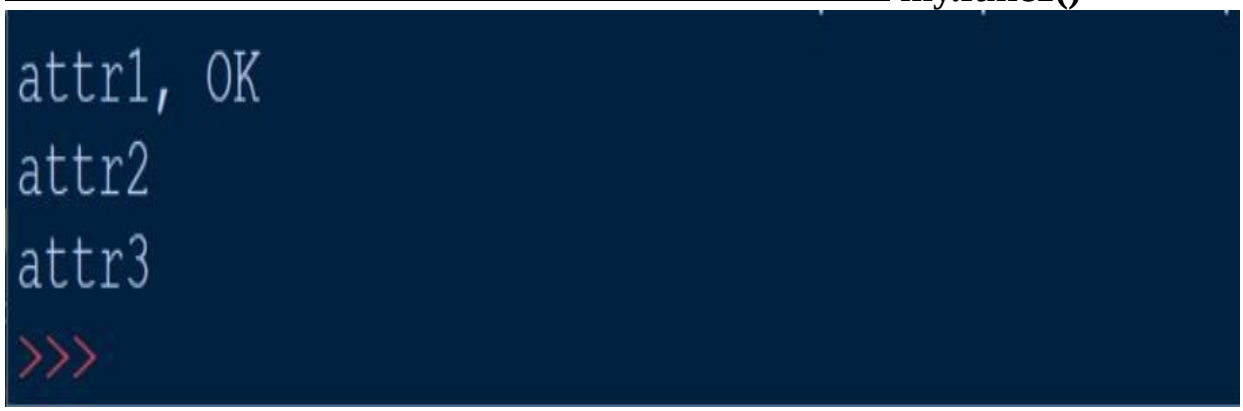
my_attr3 = 'attr3'

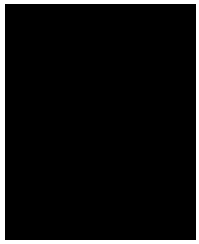
def func1(self):
print(self. my_attr1) self. func2()

self. func3()

def func2(self):
print(self. my_attr2)
def func3(self):
print(self. my_attr3)
my = my_class()
Implementation
my. my_attr1 += ', OK'

my.func1()





Checks for an object inside the object presence test and presence

test what follows has been done by function a function inside the object this way we can test the existence of an object before using it

it's a to avoid errors

The code

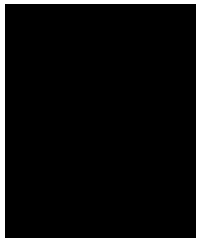
class employee:
def test():
print('Employee')

emp1 = employee()
emp2 = employee()
emp1.name = 'Ahmed'

print(hasattr(emp1, 'name')) print(hasattr(emp2, 'name')) print(hasattr(emp2, 'test'))

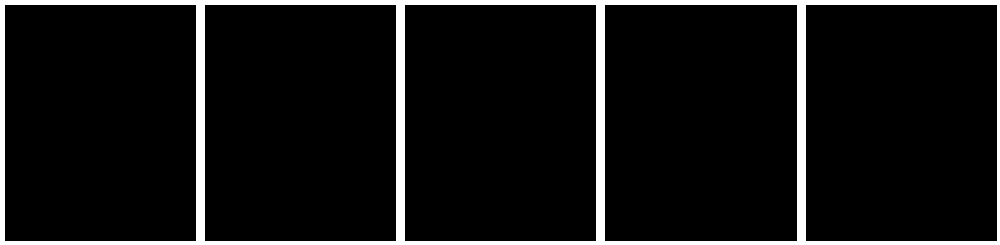
Implementation

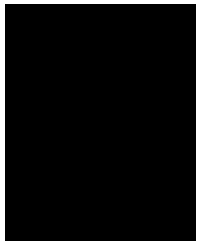
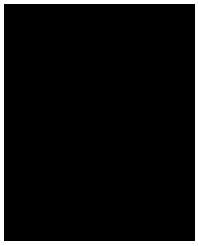
```
True
False
True
>>>
```

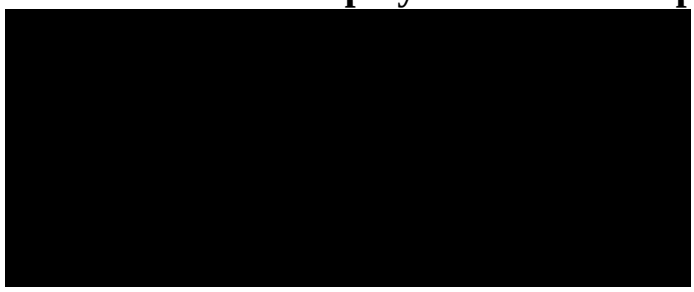


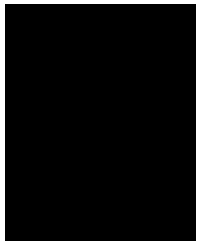
Delete properties from the object Delattr Completely omit the

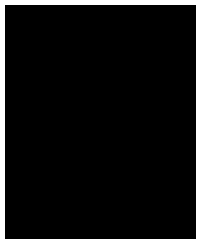
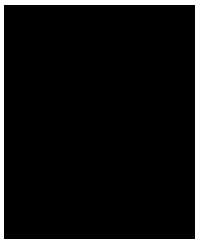
property of the function what follows has been done



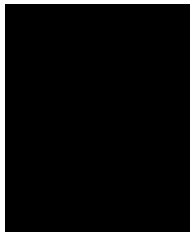
 **class employee:**  **pass**

 **emp = employee()**

  **emp.name = 'Ahmed'**

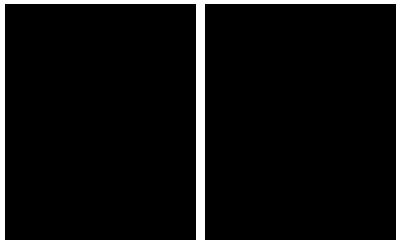
 

print(hasattr(emp, 'name'))



delattr(emp, 'name')

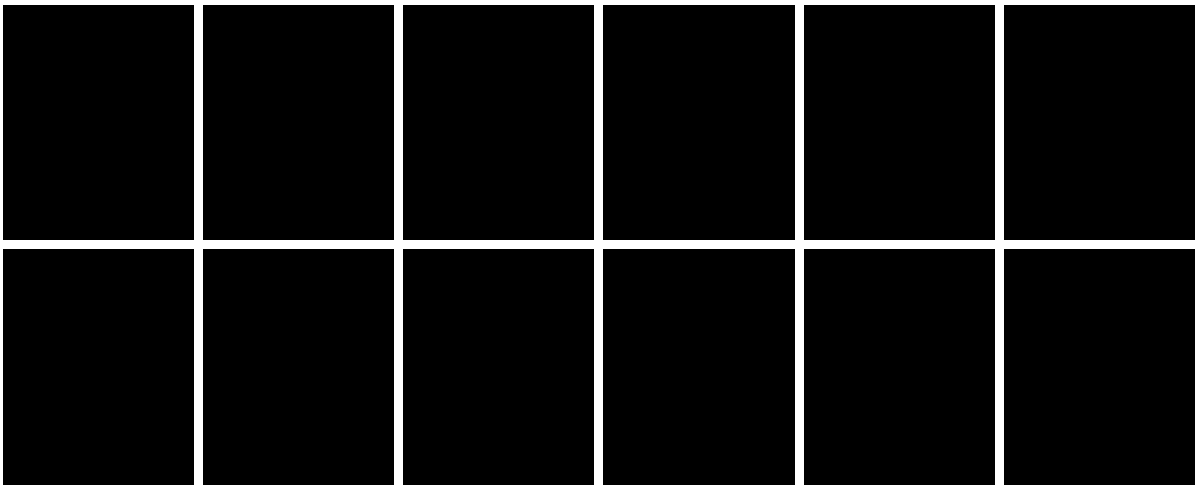
print(hasattr(emp, 'name'))



Implementation

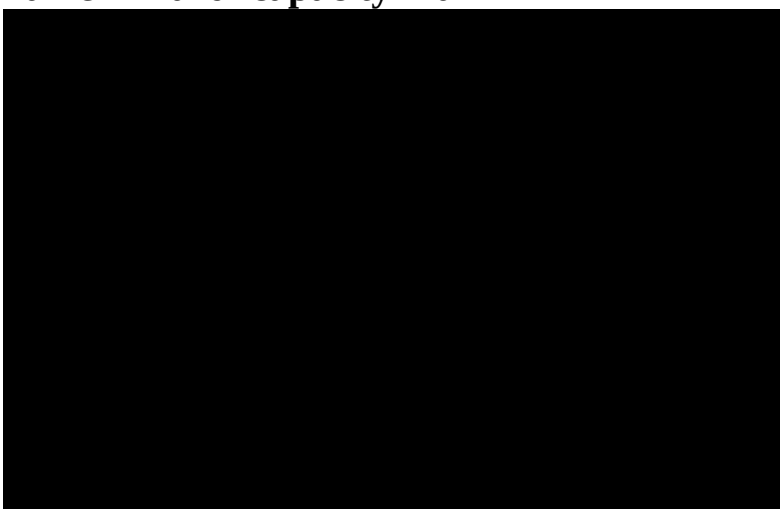
Class inner inline class and other class inside what follows has been done the basic callus can be used and the class can be used

it's a branching from it



Implementation

```
ram
pc
ram
hard
>>>
```

class computer:  **name = 'pc' generation = 5**
class hard:
name = 'hard' capacity = 0
 **speed = 0**
class ram:

name = 'ram' ramtype = 'ram' size = 0

r1 =

computer.ram()
print(r1.name)
com1 = computer()

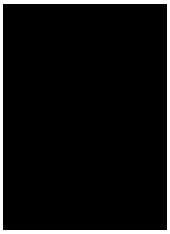
print(com1.name) print(com1.ram.name) print(com1.hard.name)

Heredity Class Person that here is the process of inheritance

Heredity Class Person that here is the process of inheritance between class he inherited everything in the employee where the

between class he inherited everything in the employee where the

employee after definition person and that by parentheses out



person

The code

class person:

name = 'Person'

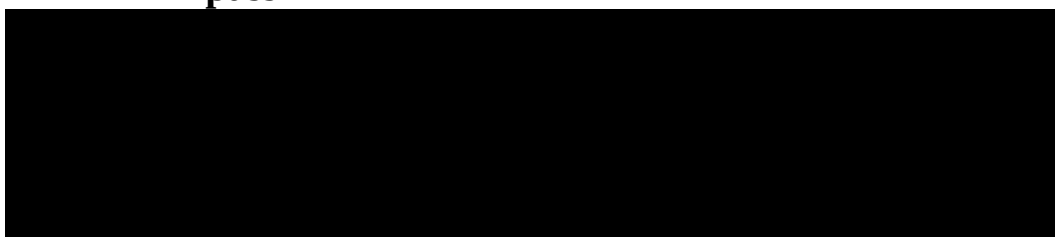
address = 'Egypt'

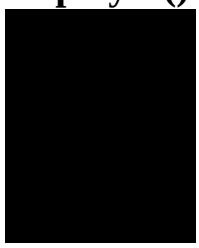
def printdata(self):

print(self.name + ' ; ' + self.address)

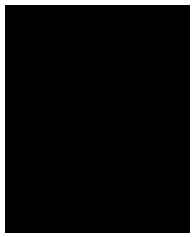
 **class employee(person):**

 **pass**

 **emp1 = employee()**



print(emp1.name)



print(emp1.address)

print('=============')

emp1.printdata()

Implementation


```
Person
Egypt
============
Person ; Egypt
>>>
```

multiple inheritance other data from the person inherit the what

follows has been done so it became hereditary person from the employee then the


follows has been done so it became hereditary person from the


employee then the

The code


The code

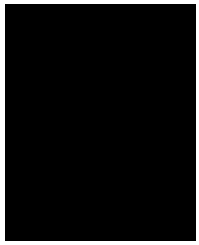
class otherdata:

email = 'example@domain.com'

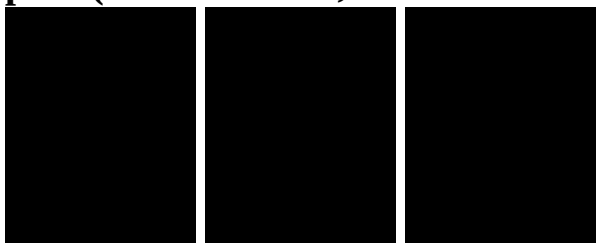
phone = '000000'

class person(otherdata): name = 'Person' address = 'Egypt'


def printdata(self):

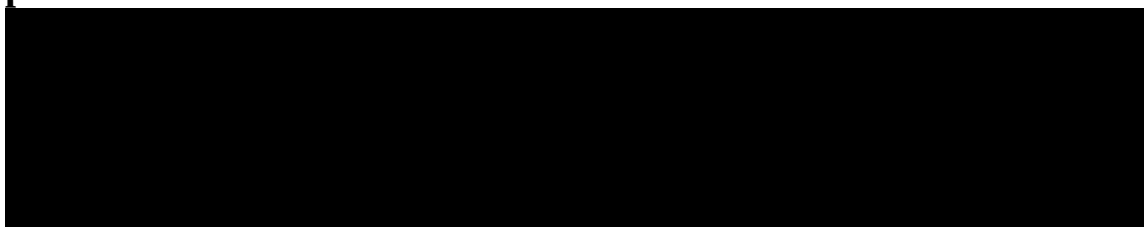


print(self.name + ' ; ' + self.address)



class employee(person):

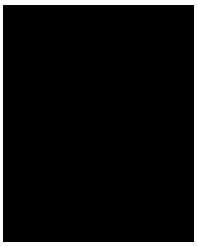
pass



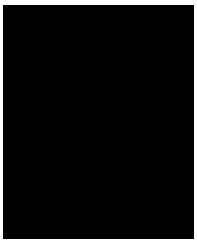
emp1 = employee()

print(emp1.email)

print(emp1.phone)



Implementation

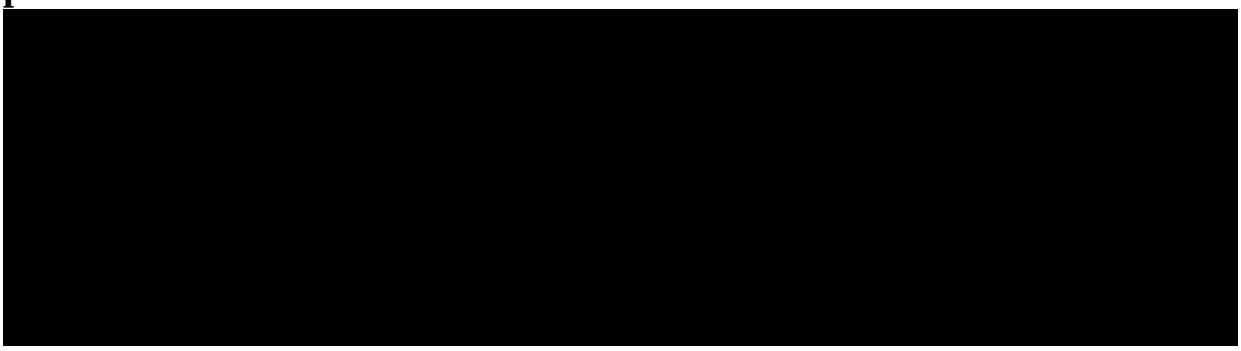


Employee do multiple inheritance within parentheses of the

It was done

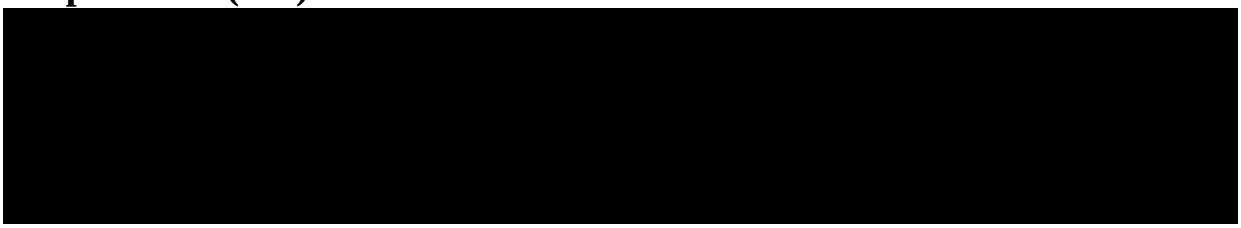
The code

class otherdata:
email = 'example@domain.com'
phone = '000000'



class person():
name = 'Person'

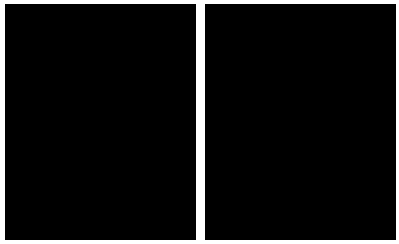
```
address =
'Egypt'
def printdata(self):
```



```
print( self.name + ' ; ' + self.address )
class employee( person , otherdata):

pass
```

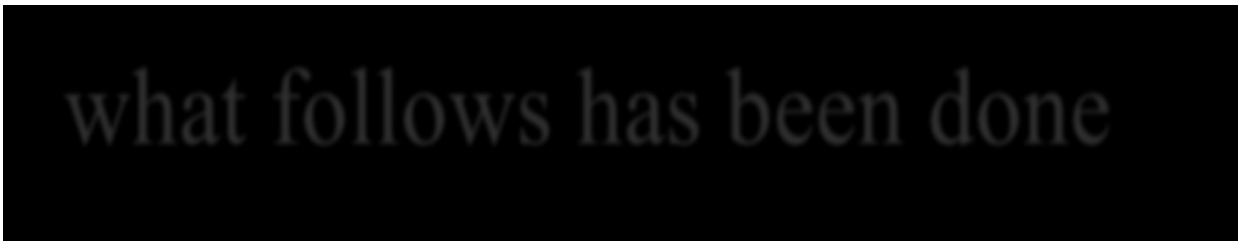
```
emp1 = employee()
print( emp1.email )
```

print(emp1.phone)



Implementation

Show documentation of objects use doc to show documentation what follows has been done



class person:
'''This is person class
This for Employee or docto or student
'''

class employee:
'This is employee class'

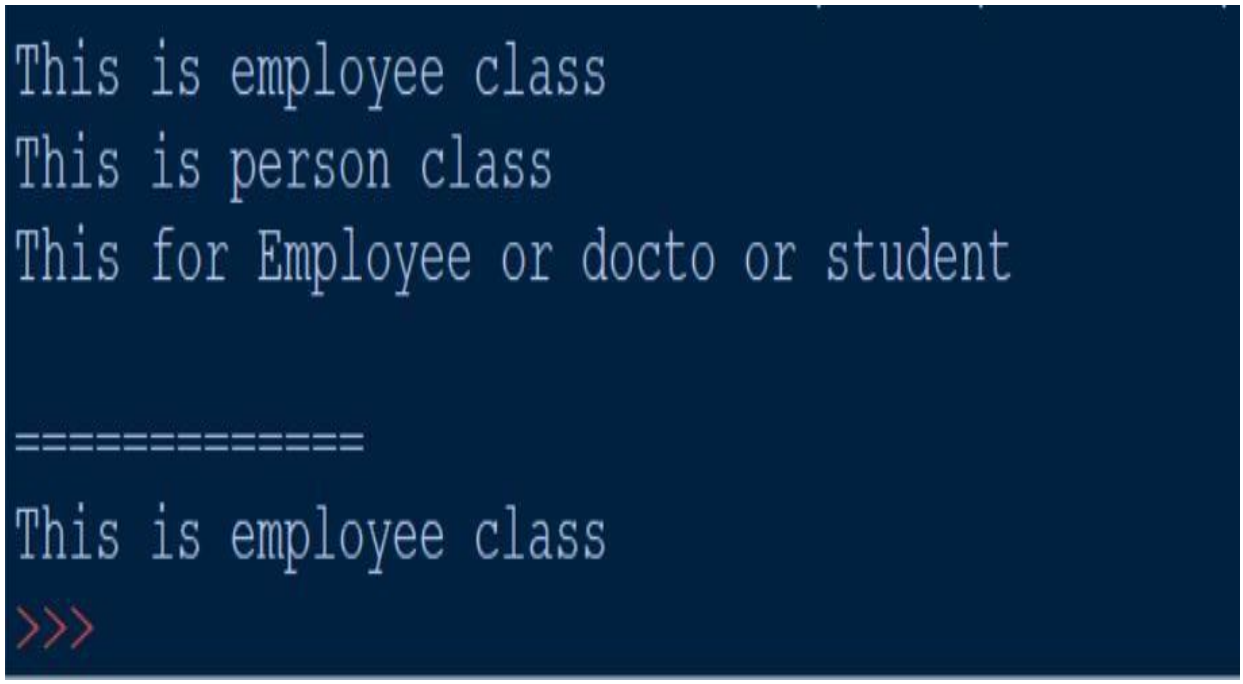
print(employee. doc)
print(person. doc)

print('==========')


Implementation

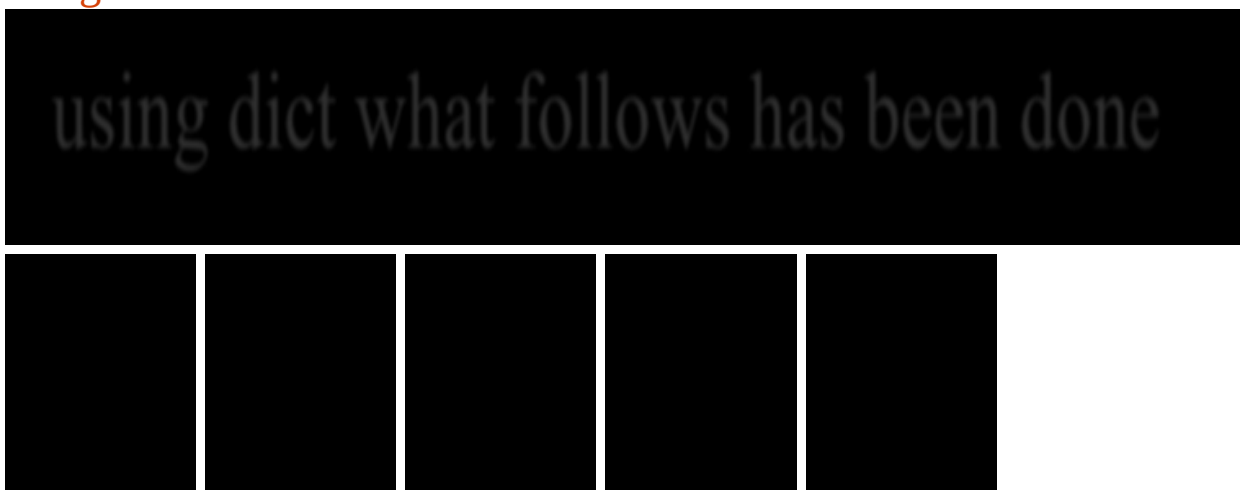
emp = employee()

print(emp. doc)



Reveal the dictionary of things find out everything in anything

using dict what follows has been done

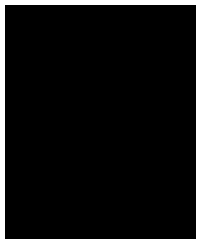


class employee:
'This is employee class' name = 'empty'
def printname(self):

print(self.name) print(employee. dict) print('================')
emp = employee()
emp.city = 'Cairo'
print(emp. dict)

Implementation

```
{'__module__': '__main__', '__doc__': 'This is employee class', 'name': 'empty', 'printname': <function employee.printname at 0x0000021250C32B70>, '__dict__': <attribute '__dict__' of 'employee' objects>, '__weakref__': <attribute '__weakref__' of 'employee' objects>}
==============
{'city': 'Cairo'}
>>>
```



Show the name of a Class

The code

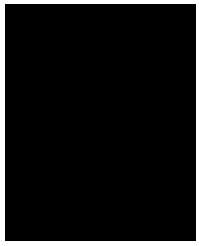
class employee: pass
class doctor: pass
class computer:

class hard: pass
class student: pass

print(employee. name)
print(doctor. name)
print(computer. name

)
print(computer.hard. name)
print(student. name)

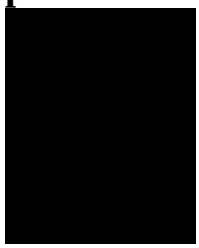
Implementation



Show the module for Class And put the following code inside

my.py First, create a file

class person:
pass



Second, apply the following to the main implementation file test.py

Use the module to display the module

What follows has been done

The code

import my
class employee:
pass
print(employee. module
print(my.person. name
)

)
print(my.person. module
)

Implementation

Show the parent class Use base to show the class created What follows has been done The process of inheriting it automatically

object inheritance from the What follows has been done

The code

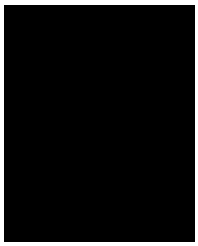
class other:


pass


class person:


name = ''

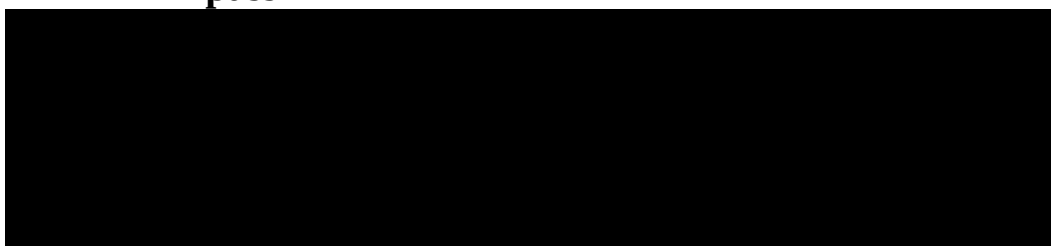

address = ''



class employee(person , other):


pass

class doctor(employee):

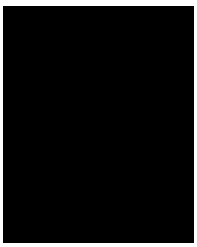
pass



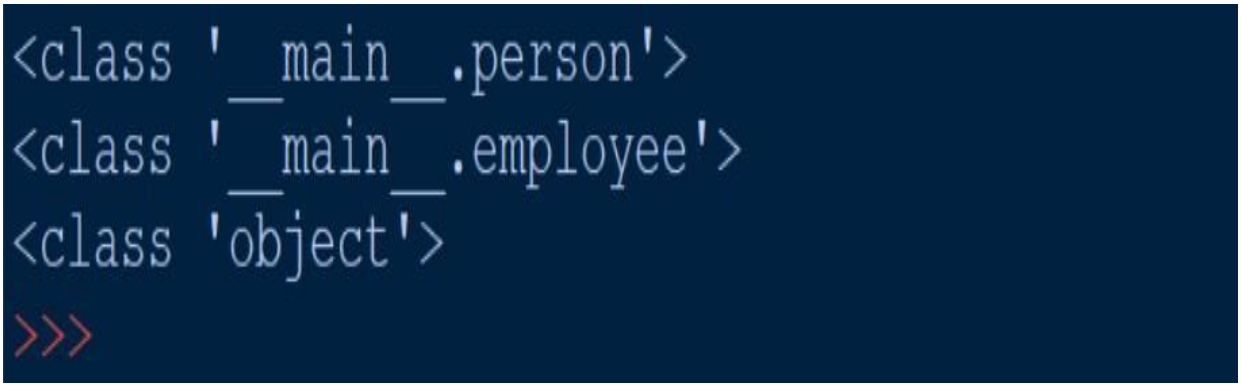
print(employee. base)

print(doctor. base)

print(person. base)



Implementation



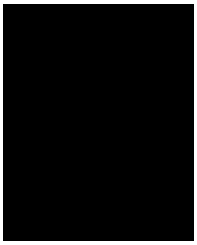
Show all parents' dials to show inspect from getmro use what

follows has been done The life story of a Class

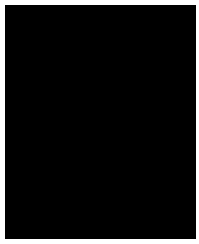
The code


```
class other:pass
class person:pass
class employee(person , other):pass

import inspect
print( inspect.getmro(employee) )
```

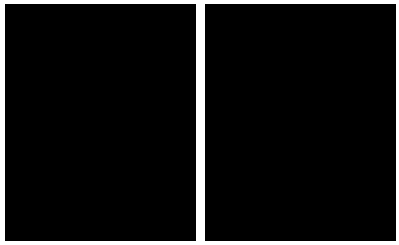
Implementation



```
(<class '__main__.employee'>, <class '__main__
_.person'>, <class '__main__.other'>, <class
'object'>)
>>>
```


Use bases to show all dials what follows has been done

that were inherited from


The code

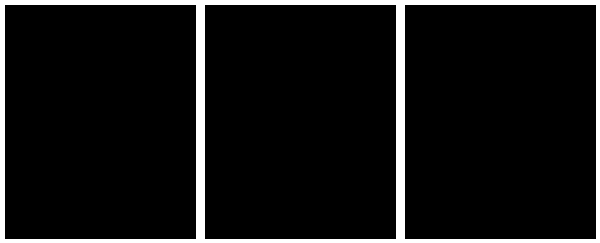
class otherdata2:pass class otherdata: email = ''
phone = ''

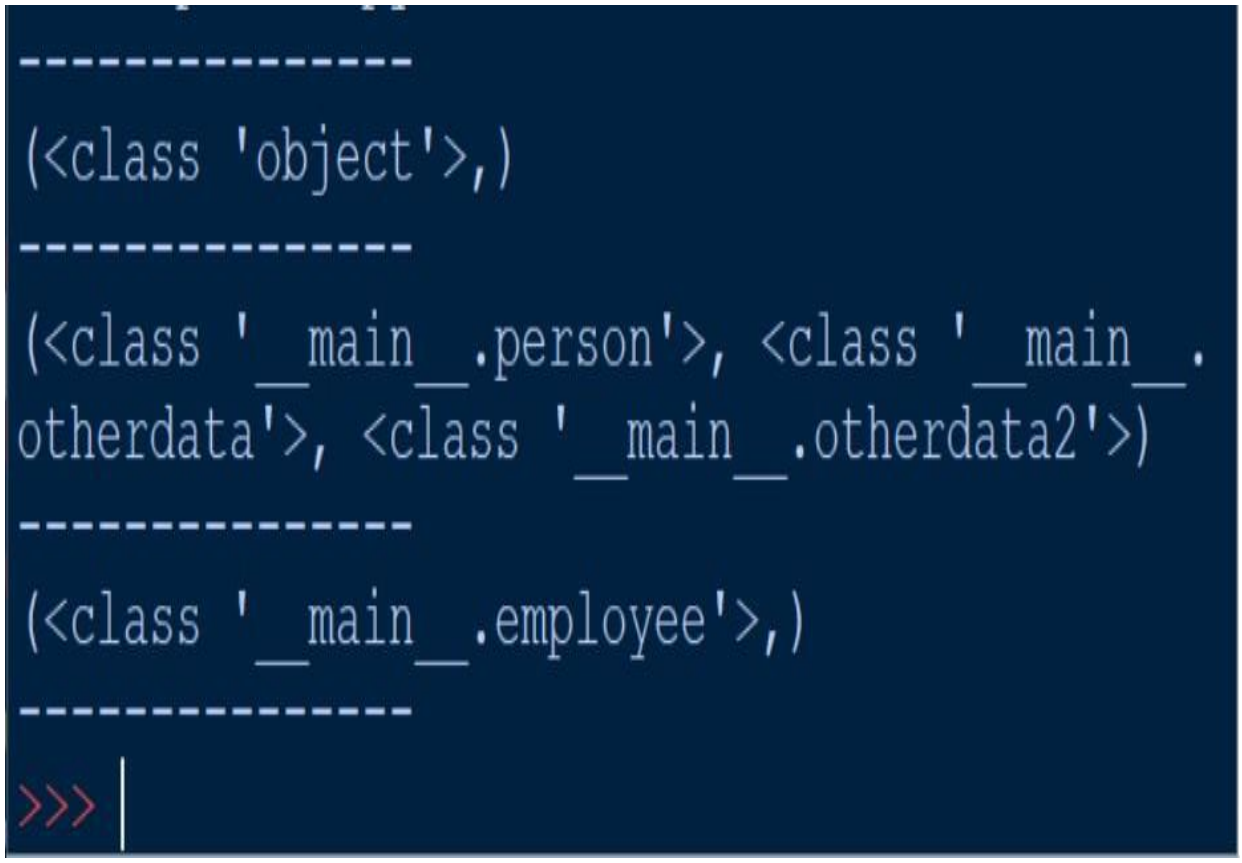
class person: name = '' address = ''

class employee(person , otherdata , otherdata2): employeeid = 0
class doctor(employee):pass

print('---------------')
print(otherdata. bases)
print('---------------')
print(employee. bases)
print('---------------')
print(doctor. bases)
print('---------------')

Implementation






129

Rewrite of functions rewrite of functions by writing the function

Rewrite of functions rewrite of functions by writing the function

what follows has been done once again in the callus that he

what follows has been done once again in the callus that he

inherited to be dedicated to him

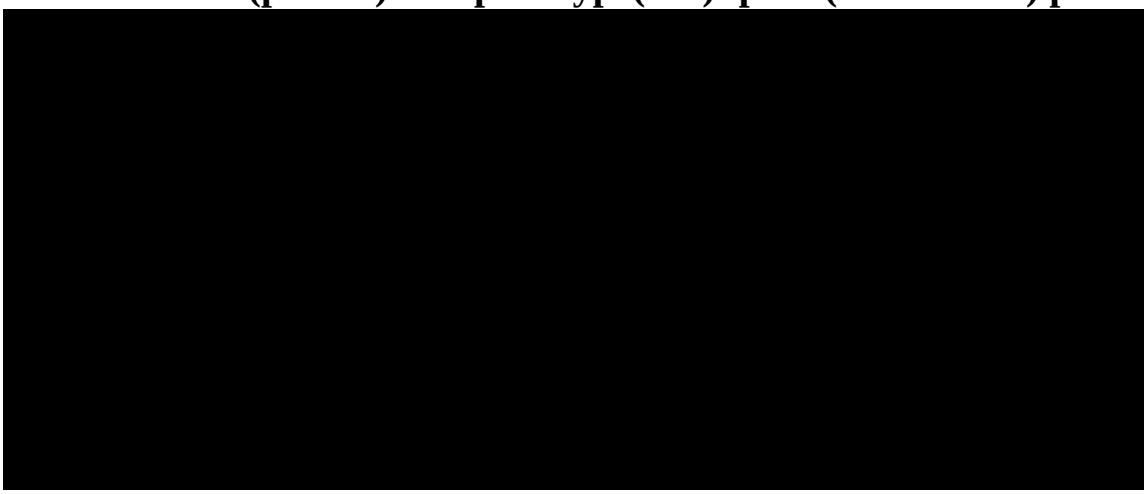
The code

class person:

def printtype(self):

print('Person')

class customer(person): def printtype(self): print('Customer') pass



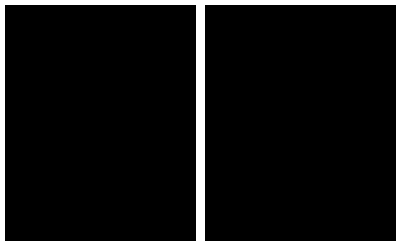
class employee(person): def printtype(self): print('Employee') pass

p = person()

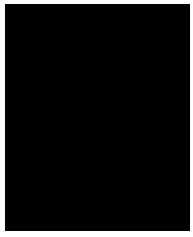
c = customer()

e = employee()

d = doctor()

 **p.printtype()**

c.printtype()



e.printtype()

d.printtype() class doctor(employee):

 **def printtype(self): print('Doctor') pass**

[redacted]

```
Person
Customer
Employee
Doctor
>>>
```

[redacted] Show the name of the object class knowing the name of the class for any object by means of what follows has been done

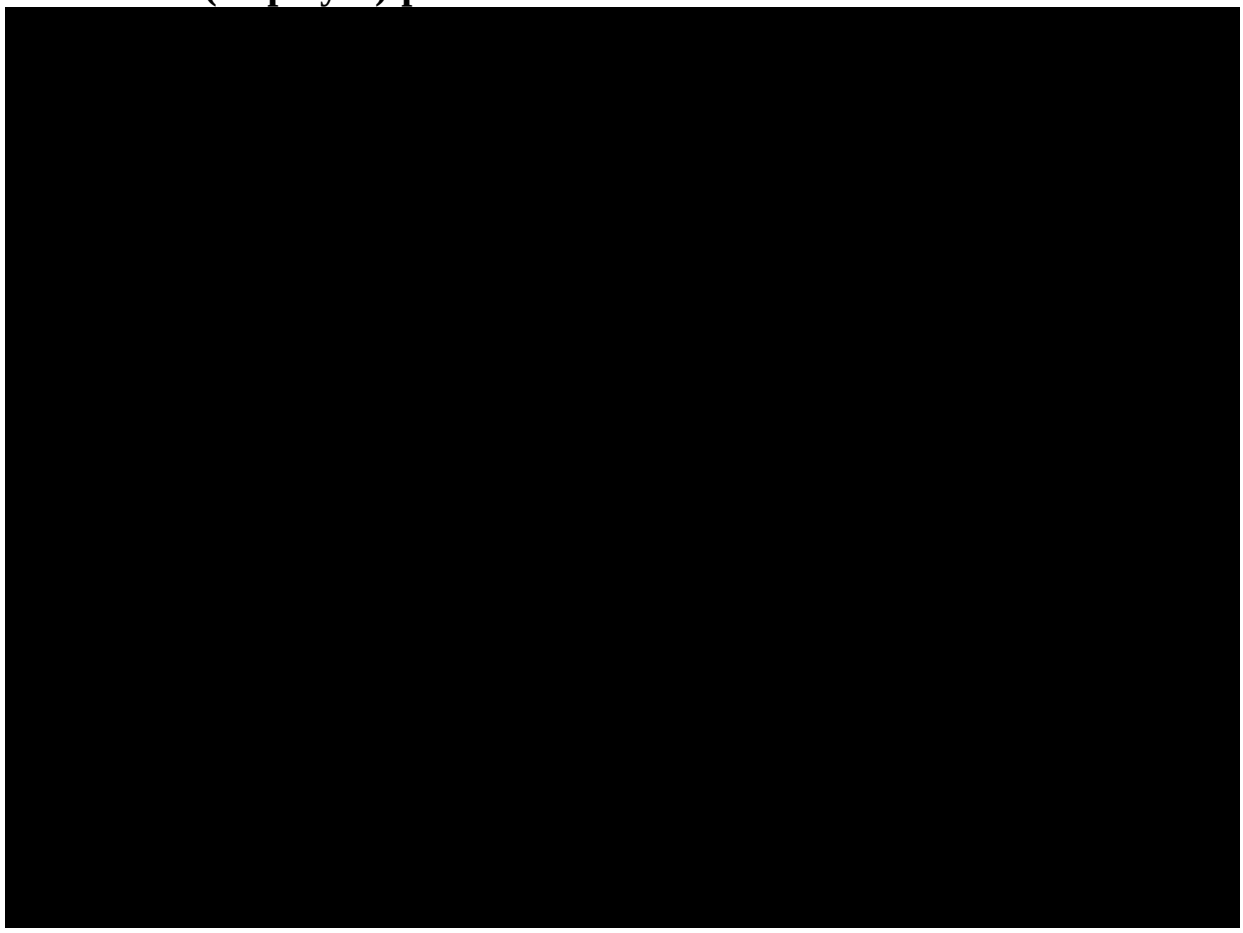
[redacted]

Easily name then class

[redacted]

The code

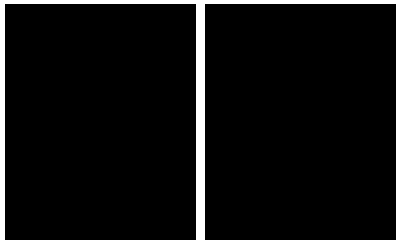
class person:
def printtype(self):
print(self. class . name)
class customer(person):pass
class employee(person):pass
class doctor(employee):pass




p = person()

c = customer()
e = employee()

d = doctor()


p.printtype()

c.printtype()

e.printtype()

d.printtype()

```
person
customer
employee
doctor
>>>
```

THANK YOU