

What is python?

Python is a high-level, interpreted programming language that is widely used for web development, scientific computing, data analysis, artificial intelligence, and many other applications.

Python is known for its simplicity, readability, and ease of use. Its syntax is straightforward and easy to learn, making it an excellent language for beginners. At the same time, it is powerful enough to handle complex projects and has an extensive library of modules and tools that make programming more efficient.

Python is an interpreted language, meaning that the code is not compiled before execution. Instead, the code is interpreted line-by-line at runtime, which allows for faster development and easier debugging.

What is python Used For ?

Python is a versatile programming language that can be used for a wide range of applications. Here are some of the most common use cases for Python:

1. Web Development.
2. Data Science.
3. Machine Learning.
4. Scripting and Automation.
5. Scientific Computing.
6. Game Development.
7. Artificial Intelligence.
8. Internet of Things (IoT).

IDLE

The Python programming language has a wide range of syntactical constructions, standard library functions, and Interactive DeveLopment Environment features.

IDLE is a simple integrated development environment (IDE) that comes with Python. It's a program that allows you to type in your programs and run them.

While the Python interpreter is the software that runs your Python programs, the interactive development environment (IDLE) software is where you'll enter your programs.

Typing things in python

Case Case matters. To Python, `print`, `Print`, and `PRINT` are all different things.

Spaces Spaces matter at the beginning of lines, but not elsewhere.

For example, the code below will not work.

```
no = eval (input('Enter Number: '))
```

On the other hand, spaces in most other places don't matter. For instance, the following lines have the same effect:

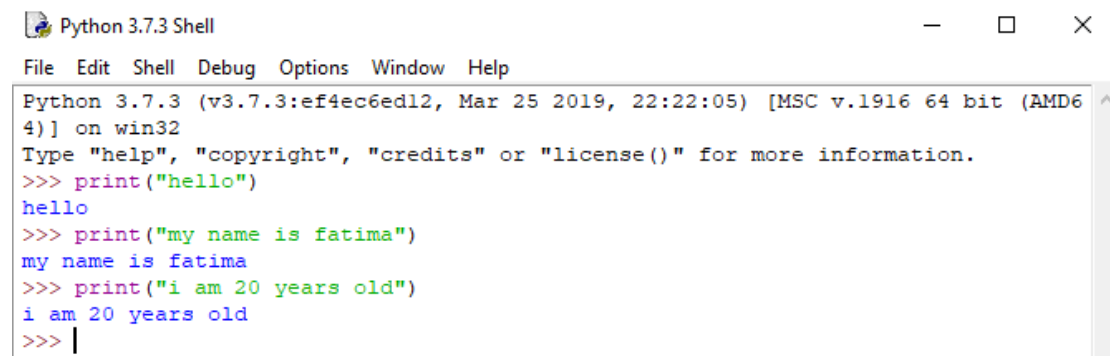
```
print('Hello world!')
print ('Hello world!')
print( 'Hello world!' )
```

Who to print any thing in python?

```
Print("hello ")
```

```
Print("my name is Fatima ")
```

```
Print("I am 20 years old")
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("hello")
hello
>>> print("my name is fatima")
my name is fatima
>>> print("i am 20 years old")
i am 20 years old
>>> |
```

If we want to change the names and ages and other information above we should use variables to make it easy.

Like this :

```
name="Fatima"
age="22"
print("my name is "+name )
print(" I am" +age + "years old ")
```

Your First Program

While the interactive shell is good for running Python instructions one at a time, to write entire Python programs, you'll type the instructions into the file editor. The file editor is similar to text editors such as Notepad or TextMate, but it has some specific features for typing in source code. To open the file editor in IDLE, select File=>New Window.

Now it's time to create your first program! When the file editor window opens, type the following into it:

```
# This program says hello and asks for my name.  
print('What is your name?')  
myName = input()  
print('It is good to meet you, ' + myName)  
print('The length of your name is:')  
print(len(myName))  
print('What is your age?')  
myAge = input()  
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

You should **save** your programs every once in a while as you type them.

To run program ,Select **Run=>Run Module**

The program's output in the interactive shell should look something like this: Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32 Type "copyright", "credits" or "license()" for more information. >>>

===== RESTART =====

>>> What is your name?

Ali

It is good to meet you, Ali

The length of your name is: 3

What is your age? 21

You will be 22 in a year.

>>>

To reload a saved program, select **File=>Open** from the menu. In the window that appears, choose the program you want to open , and click the Open button. The program should open in the file editor window.

With your new program open in the file editor, let's take a quick tour of the Python instructions it uses by looking at what each line of code does.

Comments

The following line is called a comment.

```
# This program says hello and asks for my name.
```

Python ignores comments, and you can use them to write notes or remind yourself what the code is trying to do.

The **Input()** function is a simple way for your program to get information from people using your program.

Example1:

```
name = input('Enter your name: ')
print('Hello, ', name)
```

The basic structure is

variable name = input(message to user)

The above works for getting text from the user. To get numbers from the user to use in calculations, we need to do something extra.

Example2:

```
num = eval(input('Enter a number: '))
print('Your number squared:', num*num)
```

Example3:

```
temp = eval(input('Enter a temperature in Celsius: '))
print('In Fahrenheit, that is', 9/5*temp+32)
```

The **eval** function converts the text entered by the user into a number.

The **Print()** function

The print function requires parenthesis around its arguments. Anything inside quotes will (with a few exceptions) be printed exactly as it appears. In the following, the first statement will output 3+4, while the second will output 7.

print('3+4')

print(3+4)

To print several things at once, separate them by commas.

```
print('The value of 3+4 is', 3+4)
print('A', 1, 'XYZ', 2)
```

```
The value of 3+4 is 7
A 1 XYZ 2
```

Optional arguments

sep Python will insert a space between each of the arguments of the print function. There is an optional argument called sep, short for separator, that you can use to change that space to some thing else.

```
print('the value of 3+4=',3+4,'.',sep=' ')
```

the value of 3+4= 7 .

end The print function will automatically advance to the next line.

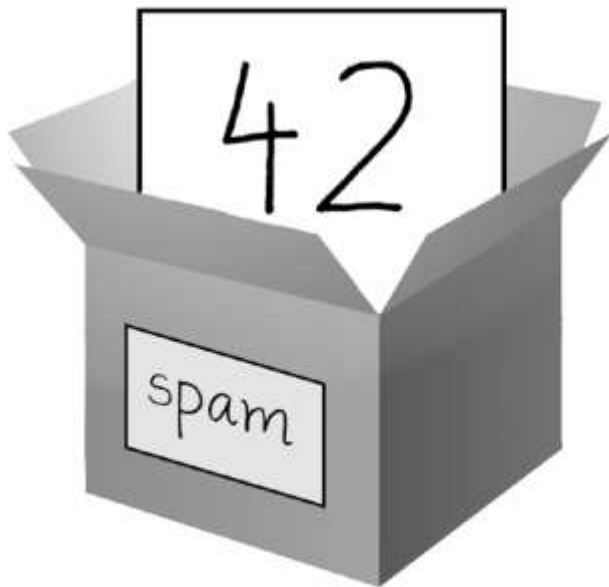
```
print('On the first line', end='')  
print('On the second line')
```

On the first lineOn the second line

Variables

A variable is like a box in the computer's memory where you can store a single value. You'll store values in variables with an assignment statement.

Think of a variable as a labeled box that a value is placed in, spam=42:



*Figure 1-2: spam = 42 is like telling the program,
"The variable spam now has the integer value 42 in it."*

what the values of x and y will be after the code is executed.

```
x=3  
y=4  
z=x+y  
z=z+1  
x=y  
y=5
```

After these four lines of code are executed, x is 4, y is 5 and z is 8.

Variable names

There are just a couple of rules to follow when naming your variables.

- Variable names can contain letters, numbers, and the underscore.
- Variable names cannot contain spaces.
- Variable names cannot start with a number.
- Case matters—for instance, temp and Temp are different.

Valid variable names	Invalid variable names
balance	current-balance (hyphens are not allowed)
currentBalance	current balance (spaces are not allowed)
current_balance	4account (can't begin with a number)
_spam	42 (can't begin with a number)
SPAM	total_\$um (special characters like \$ are not allowed)
account4	'hello' (special characters like ' are not allowed)

Reserved Words

- You cannot use **reserved words** as variable names / identifiers

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield

EX1: Write a program that asks the user to enter three numbers (use three separate input statements). Create variables called **total** and **average** that hold the sum and average of the three numbers and print out the values of total and average.

```
#this program to find total and avarege
n1=eval(input('enter no1'))
n2=eval(input('enter no2'))
n3=eval(input('enter no3'))
total=n1+n2+n3
average=total/3
print("total= ",total,"average= ",average)
```

Ex2:

Ask the user to enter a number. Print out the square of the number, but use the `sep` optional argument to print it out in a full sentence that ends in a period. Sample output is shown below.

```
Enter a number: 5
The square of 5 is 25.
```

Numeric Expressions

```
>>> xx = 2  /\
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
```

```

      4 R 3
5 | 23
   20
   --
    3
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right



```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
```

Parenthesis
Power
Multiplication
Addition
Left to Right



1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

1 + 2 * 5

1 + 10

11

Several Types of Numbers:

- Numbers have two main types
 - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
 - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> x=1
>>> type(x)
<class 'int'>
>>> t=89.6
>>> type(t)
<class 'float'>
>>> type(1)
<class 'int'>
>>> type(10.5)
<class 'float'>
```


The str(), int(), and float() Functions

The str(), int(), and float() functions will evaluate to the string, integer, and floating-point forms of the value you pass, respectively. Try converting some values in the interactive shell with these functions, and watch what happens.

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

What does the variable bacon contain after the following code runs?

```
bacon = 20
bacon + 1
```

Why does this expression cause an error? How can you fix it?

```
'I have eaten ' + 99 + ' burritos.'
```

Math functions

The math module Python has a module called math that contains familiar math functions, including sin, cos, tan, exp, log, log10, factorial, sqrt, floor, and ceil. There are also the inverse trig functions, hyperbolic functions, and the constants pi and e. Here is a short example:

```
from math import sin, pi
print('Pi is roughly', pi)
print('sin(0) =', sin(0))
```

```
Pi is roughly 3.14159265359
sin(0) = 0.0
```

Strings

- A string object is a 'sequence', i.e., it's a list of items where each item has a defined position.
- Each character in the string can be referred, retrieved and modified by using its position.
- This order is called the 'index' and always starts with 0.

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example –

```
>>> var1='Hello world'
>>> var2="python programming"
>>> print('var1[0]= ',var1[0])
var1[0]= H
>>> print('var2[1:5]',var2[1:5])
var2[1:5] ytho
```

```
>>> s='Hello'
>>> z='world'
>>> print(s+z)
Helloworld
>>> print(s+' '+z)
Hello world
>>> print (s+z*4)
Helloworldworldworldworld
>>> print(s+ 'world '*4)
Helloworld world world world
```

Updating Strings

You can "update" an existing string by reassigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. For example –

```
#!/usr/bin/python
var1 = 'Hello World!'
print "Updated String :- ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result –

```
Updated String :- Hello Python
```

String Methods

Python includes the following built-in methods to manipulate strings

capitalize

Capitalizes first letter of string

```
>>> print(var1.capitalize())
Hello world
```

isalnum

Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

isalpha

Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.

isnumeric

Returns true if a unicode string contains only numeric characters and false otherwise.

isupper

Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

islower

Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.

lenstring

Returns the length of the string

lower

Converts all uppercase letters in string to lowercase.

upper

Converts lowercase letters in string to uppercase.

replaceold, new[, max]

Replaces all occurrences of old in string with new or at most max occurrences if max given.

```
>>> s='Awatif Ali'
>>> print(s.replace('Ali', 'Salman'))
Awatif Salman
```

```
>>> var1='mathmatic'
>>> print(var1.capitalize())
Mathmatic
>>> print(var1.upper())
MATHMATIC
>>> print(var1.isalpha())
True
>>> print(var1.isalpha())
True
>>> print(var1.isnumeric())
False
>>> print(var1.lower())
mathmatic
>>> print(var1.isalnum())
True
```

(\n): it used to get new line .

(\") : it used to put a quotation mark.

(\): is used to put a slash .

(\t) : it use to put a space after the word .

```
>>> print('Mathematic')
Mathematic
>>> print('math\nemat\n tic')
math
emat
tic
>>> print('math\" ematic')
math" ematic
>>> print('math \ ematic')
math \ ematic
>>> print('math \t ematic')
math      ematic
```

We will often want to pick out individual characters from a string. Python uses square brackets to do this. The table below gives some examples of indexing the string `s='Python'`.

Statement	Result	Description
<code>s[0]</code>	P	first character of <code>s</code>
<code>s[1]</code>	y	second character of <code>s</code>
<code>s[-1]</code>	n	last character of <code>s</code>
<code>s[-2]</code>	o	second-to-last character of <code>s</code>

A *slice* is used to pick out part of a string. It behaves like a combination of indexing and the `range` function. Below we have some examples with the string `s='abcdefghij'`.

```
index:      0 1 2 3 4 5 6 7 8 9
letters:    a b c d e f g h i j
```

Code	Result	Description
<code>s[2:5]</code>	cde	characters at indices 2, 3, 4
<code>s[:5]</code>	abcde	first five characters
<code>s[5:]</code>	fg hij	characters from index 5 to the end
<code>s[-2:]</code>	ij	last two characters
<code>s[:]</code>	abcdefghij	entire string
<code>s[1:7:2]</code>	bdf	characters from index 1 to 6, by twos
<code>s[: :-1]</code>	jihgfedcba	a negative step reverses the string

If statements:

Comparison Operators

Comparison operators compare two values and evaluate down to a single Boolean value. Table 2-1 lists the comparison operators.

Table 2-1: Comparison Operators

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

These operators evaluate to True or False depending on the values you give them. Let's try some operators now, starting with == and !=.

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True

>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
>>> 42 == '42'
False
```

Boolean Operators

The three Boolean operators (and, or, and not) are used to compare Boolean values. Like comparison operators, they evaluate these expressions down to a Boolean value. Let's explore these operators in detail, starting with the and operator.

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

EX:

```
grade = eval(input('Enter your score: '))
if grade >= 90:
    print('A')
if grade >= 80 and grade < 90:
    print('B')
if grade >= 70 and grade < 80:
    print('C')
if grade >= 60 and grade < 70:
    print('D')
if grade < 60:
    print('F')
```

Else

The **else** keyword catches anything which isn't caught by the preceding conditions.

Write a program in which the user enters a number and test whether this number is even or odd ?

```
x=int(input("Please Enter The Number : "))
if x%2 ==0:
    print("is even ")
else :
    print( "is odd ")
```

write a program in which the user input any word and the program check if this word is uppercase or lowercase ?

```
a=input("Please enter any word : " )
if a.isupper() :
    print(a + " is uppercase ")
else :
    print(a + " is lowercase")
```

Elif

The **elif** keyword is python's way of saying "if the previous conditions were not true, then try this condition".

```
grade = eval(input('Enter your score: '))

if grade >= 90:
    print('A')
elif grade >= 80:
    print('B')
elif grade >= 70:
    print('C')
elif grade >= 60:
    print('D')
else:
    print('F')
```


Python Loops

Python has two primitive loop commands:

- **while** loops
- **for** loops

The **while** Loop

With the while loop we can execute a set of statements as long as a condition is true. You can make a block of code execute over and over again with a while statement.

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

You can see that a **while** statement looks similar to an **if** statement.

Let's look at an if statement and a while loop that use the same condition and take the same actions based on that condition.

Here is the code with an **if** statement:

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

يطبع

Hello, world.

Here is the code with a **while** statement:

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

يطبع

Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.

The **break** Statement

There is a shortcut to getting the program execution to break out of a while loop's clause early. If the execution reaches a break statement, it immediately exits the while loop's clause. In code, a break statement simply contains the break keyword.

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

continue Statements

Like break statements, continue statements are used inside loops. When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and reevaluates the loop's condition.

Example

Continue to the next iteration if i is 3:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        continue
    i += 1
```

For Loops

A For loop is used for iterating over a sequence.

Example 1:

```
for i in range(1, 10):  
    print(i)
```

Example 2: This program gets 10 numbers from the user and counts how many of those numbers are greater than 10.

```
count = 0  
for i in range(10):  
    num = eval(input('Enter a number: '))  
    if num>10:  
        count=count+1  
print('There are', count, 'numbers greater than 10.')
```

Example 3: This modification of the previous example counts how many of the numbers the user enters are greater than 10 and also how many are equal to 0. To count two things we use two count variables.

```
count1 = 0  
count2 = 0  
for i in range(10):  
    num = eval(input('Enter a number: '))  
    if num>10:  
        count1=count1+1  
    if num==0:  
        count2=count2+1  
print('There are', count1, 'numbers greater than 10.')
```

Example 4: This program will add up the numbers from 1 to 100. The way this works is that each time we encounter a new number, we add it to our running total, s.

```
s = 0  
for i in range(1,101):  
    s = s + i  
print('The sum is', s)
```

Example 5: This program that will ask the user for 10 numbers and then computes their average.

```
s = 0  
for i in range(10):  
    num = eval(input('Enter a number: '))  
    s = s + num  
print('The average is', s/10)
```

Example 4 We can use a **while** loop to mimic a for loop, as shown below. Both loops have the exact same effect.

```
for i in range(10):  
    print(i)  
  
i=0  
while i<10:  
    print(i)  
    i=i+1
```

Lists

Like a string, a *list* is a sequence of values. In a string, the values are characters; in a list, they can be any type. The values in list are called *elements* or sometimes *items*.

There are several ways to create a new list; the simplest is to enclose the elements in square brackets ("[" and "]"):

```
[10, 20, 30, 40]  
['crunchy frog', 'ram bladder', 'lark vomit']
```

The first example is a list of four integers. The second is a list of three strings. The elements of a list don't have to be the same type. The following list contains a string, a float, an integer, and (lo!) another list:

```
['spam', 2.0, 5, [10, 20]]
```

A list within another list is *nested*.

A list that contains no elements is called an empty list; you can create one with empty brackets, [].

As you might expect, you can assign list values to variables:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']  
>>> numbers = [17, 123]  
>>> empty = []  
>>> print(cheeses, numbers, empty)  
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

Lists are mutable

The syntax for accessing the elements of a list is the same as for accessing the characters of a string: the bracket operator. The expression inside the brackets specifies the index. Remember that the indices start at 0:

```
>>> print(cheeses[0])
Cheddar
```

Unlike strings, lists are mutable because you can change the order of items in a list or reassign an item in a list. When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned.

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print(numbers)
[17, 5]
```

The one-th element of `numbers`, which used to be 123, is now 5.

You can think of a list as a relationship between indices and elements. This relationship is called a *mapping*; each index “maps to” one of the elements.

List indices work the same way as string indices:

- Any integer expression can be used as an index.
- If you try to read or write an element that does not exist, you get an `IndexError`.
- If an index has a negative value, it counts backward from the end of the list.

The `in` operator also works on lists.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

Example:1

```
friends = ['Ali','Ahmed','Hussain']
```

```
for friend in friends:
```

```
    print('Hello ',friend)
```

```
print('done')
```

Example 2:

```
s='python'
mylist=['one' , 2 , 'three','four' , 5]
print(s[0])
print()
print(mylist[0])
print(len(mylist))
print(mylist[:3])
print(mylist[-1])
print(mylist[0:2])
```

Example 3:

```
large=-1
Nlist=[51,18,20,94,35,124,16]
for n in Nlist:
    if n>large:
        large=n
    print(large,n)
print('large number is = ',large)
```

Example:4

```
s='what if we went to the zoo'
k=0
for i in s:
    if i in ['o','a','e','i','u']:
        k=k+1
print('k= ',k)
```

Example:5

```
p='I love python programming'
t=0
for chr in p:
    print(chr)
    if chr != ' ':
        t=t+1
print('t= ',t)
```

Example:6

```
mylist=[]
mylist.append(5)
mylist.append(9)
mylist.append(12)
print(mylist)
mylist.insert(2,3)
print(mylist)
mylist.sort()
print(mylist)
```

Example 7: Suppose that wordlist=['gnnan', 'hanan', 'gnaat', 'ieman']

Write python program to Print all the words that start with gn

```
wordlist=['gnnan', 'hanan', 'gnaat', 'ieman']
for word in wordlist:
    if word[:2]=='gn':
```

```
print(word)
```

Example 8 :Print all three letter words

```
wordlist=['Aml' , 'Ahmad', 'ali', 'Huda', 'saja', 'sma']
```

```
for word in wordlist:
```

```
    if len(word)==3:
```

```
        print(word)
```

Example 9 :Determine number of words start with a vowel.

```
wordlist=['Aml' , 'Ahmad', 'ali', 'Huda', 'saja', 'sma']
```

```
count = 0
```

```
for word in wordlist:
```

```
    if word[0] in 'aeiou':
```

```
        count=count+1
```

```
print('number of words start with vowel= ', count)
```

Example 10 :Print all 5-letter words that start with gn and end in at.

```
wordlist=['gnnan' , 'hanan', 'gnaat', 'ieman' , 'Aml' , 'Ahmad', 'Ali'  
, 'Huda', 'Saja', 'gnnamat' ]
```

```
for word in wordlist:
```

```
    if len(word)==5 and word[:2]=='gn' and word[-2:]=='at':
```

```
        print(word)
```

example 11:

```

s=input('enter string')
for i in range(len(s)):
    if s[i]== 'a':
        print(i)

>>>
===== RESTART: D:\python program\string
enter stringAn apple a day keeps the doctor away
3
9
12
32
34

```

Example 12:

```

s=input('enter string')
d_s=''
for c in s:
    d_s = d_s + c*2
print(d_s)

>>>
===== RESTART:
enter stringHello
HHeelllloo

>>>
===== RESTART:
enter stringwelcome
wweellccoommee

```


Example 13:

```
wordlist=['gnnan' , 'hanan', 'gnaat', 'iemana' , 'Aml' , 'Ahmad', 'Ali' ]  
for word in wordlist:  
    if len(word)==3 and word[0]=='A':  
        print(word)
```