Download apache. Extract it and copy it in c:\ thereby creating a folder. c:\apache24. Go to c:\apache24\conf folder and select httpd.conf file. Open it in notepad make following changes -

1- Options Indexes FollowSymLinks ExecCGI

2- AddHnadler cgi-script .cgi.py

3- NameServer - localhost

Now create a 'test.py' with following statement -

print('Script working')

Put this file in c:\apache24\htdocs folder.

Open command prompt as administrator. Run following command -

httpd -k start

Now open http:\\localhost in browser. You should see 'It works' message. Now open http:\\localhost\test.py . This will execute the 'test.py' in browser.

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

running Flask app on Apache -

1 - let your Flask application be flask1.py

2 - in the same directory as flask1.py, create a file named application.wsgi with following content -

import sys

#Expand Python classes path with your app's path

sys.path.insert(0, "c:/miniconda3/mayankproj")

from flask1 import app as application

3 - pip install mod\_wsgi in the same environment where Flask was installed and flask application was created. Note - pip install might fail and it may ask you to install Visual Studio C++ compiler and will give you a link to download from. Download it and restart the system. Then again pip install mod\_wsgi

4- Now run mod\_wsgi-express module-config on command line. It will output some content. Copy the output.

5- In apache 24, open httpd.conf file and paste the above output as it is in the end. Also add following in httpd.conf file -

<VirtualHost\*>

ServerName localhost

WSGIScriptAlias /5000 C:/miniconda3/mayankproj/application.wsgi

<Directory C:\miniconda3\mayankproj>

Require all granted

</Directory>

</VirtualHost>

6- That's it. Now run httpd -k run command, open your browser and go to localhost/5000 to see your flask application run

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

activate webdev

pip install psycopg2

this library serves as a driver for communication between Python and PostgreSQL.

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

C - variable length array

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

how to use cygwin

C:\Users\Mayank>cd c:\cygwin\bin

c:\cygwin\bin>bash

Mayank@Mayank-PC /usr/bin

$

-------

shift+f10 for context menu

winkey+1,winkey+2, first or second item in taskbar

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

compiled languages are usually static while interpreted languages are usually dynamic. In general, static refers to things that happen at compile time, while 'dynamic' refers to things that happen at runtime.

Also, in dynamic languages, variables are stored in memory while program is running. In compiled languages, variable names exist at compile time but not at run time. The compiler chooses a location for each variable and records these locations as part of the compiled program.1 The location of a variable is called its “address”. At run time, the value of each variable is stored at its address, but the names of the variables are not stored at all.

The -c flag tells gcc to compile the program and generate machine code, but not to link it or generate an executable:

$ gcc hello.c -c

The result is a file named hello.o, where the o stands for “object code”, which is the compiled program. Object code is not executable, but it can be linked into an executable.

Also note below -

c:\C>nm hello.o

00000000 b .bss

00000000 d .data

00000000 r .eh\_frame

00000000 r .rdata

00000000 r .rdata$zzz

00000000 t .text

U \_\_\_main

00000000 T \_main

U \_printf

U \_scanf

The UNIX command nm reads an object file and generates information about the names it defines and uses. (This command works on Windows as well)

You can control how much optimization gcc does with the -O flag. By default, it does very little optimization, which can make debugging easier.

The option -O1 turns on the most common and safe optimizations. Higher numbers turn on additional optimizations that require longer compilation time.

Similar to the -c flag, the -S flag tells gcc to compile the program and generate assembly code.

Taking another step backward through the compilation process, you can use the -E flag to run the preprocessor only:

$ gcc hello.c -E

The result is the output from the preprocessor. In this example, it contains the included code from stdio.h, and all the files included from stdio.h, and all the files included from those files, and so on. On my machine, the total is more than 800 lines of code.

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

If you have one bit, you can specify one of two possibilities, usually written 0 and 1. If you have two bits, there are 4 possible combinations, 00, 01, 10, and 11. In general, if you have b bits, you can indicate one of 2^b values. A byte is 8 bits, so it can hold one of 256 values. Going in the other direction, suppose you want to store a letter of the alphabet. There are 26 letters, so how many bits do you need? With 4 bits, you can specify one of 16 values, so that’s not enough. With 5 bits, you can specify up to 32 values, so that’s enough for all the letters, with a few values left over.

In general, if you want to specify one of N values, you should choose the smallest value of b so that 2b N. Taking the log base 2 of both sides yields

b>=log2N.

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