python tutorial for new users

```
There is actually a python2 and python3 modified python2.
Since there is a lot of python2 available for download on
the web, I cover that first, then the changes in python3.
Also, python is an interpreter and may be run interactively,
yet, for this tutorial I use input files because my typing
is very bad.
```

Contents

hello_world.py simplest program

```
If you do not have python on your Linux:

sudo apt install python

Then, to also have python 3:

sudo apt install python3

Then, for many python library packages:

sudo apt-cache search python

Links to source file .py and .out output file, click to read any not shown
hello_world.py  source file
hello_world_py.out  output file

# hello_world.py  example file    # starts comment, like // in C and Java
print "hello from python"  # three ways to print the same line
print 'hello from python'  # no extra spaces allowed at beginning of line
print "hello",             # comma at end  prevents new line
print "from python"        # quote or apostrophe same

hello_world_py.out:
hello from python
hello from python
hello from python


command line to run and output to file:

python hello_world.py > hello_world_py.out
```

A more professional structure that is more like Java:
mainex.py source file
mainex_py.out output file

```
#mainex.py  showing professional structure
def main(): # note  :  now need indentation
  print "main.py running "
  funct()                  # just calling function below
  print "note indentation and unindent" # blank line critical

def funct():
  print "in funct" # another blank line needed

if __name__ == '__main__':  # like C main program
  main()
```

mainex_py.out:

```
main.py running
in funct
note indentation and unindent
```

data type is defined by use

Subscripts in Python are the same as C and Java.
[0] is first, [1] is second [n-1] is last for n items.

Links to source file and output, shown below
hello.py source file
hello_py.out output file

```
# hello.py  example file  indentation and blank lines important
#           simple .py can be without  def main():
#                                      if __name__ == '__main__':
#                                      main()
import math

def main():
  print "hello from python"  # three ways to print the same line
  print 'hello from python'
  print "hello",
  print "from python"
  a_int=7
  a_float=7.25
  a_real=7.2597256357e+300
  print "a_int=",
  print a_int
  print "a_float=",
  print a_float
  print "a_real=",
  print a_real

  a_list=[7, 7.25, "abc"]
  print "a_list=",
  print a_list
  print "a_list[2]=",
```

```
  print a_list[2]
  a_list.append("def")
  print "append=",
  print a_list
  del a_list[1]
  print "delete=",
  print a_list
  print "len(a_list)=",
  print len(a_list)
  print " "

  a_tuple=2, 3, 5, 7   # can not modify
  b_tuple=(2, 3, 5, 7) # same as above
  print "a_tuple=",
  print a_tuple
  print "b_tuple=",
  print b_tuple
  print "b_tuple[2]=",
  print b_tuple[2]
  print " "

  dict={1:'a', 2:'b', 3:'c'} # dictionary key : value
  print "dict=",
  print dict
  dict[5]='e'
  print "dict 5=",
  print dict
  print "dict[3]=", # look up by key
  print dict[3]
  print " "

  z=range(5)
  print "range(5)=",
  print z
  z=range(1,5)
  print "range(1,5)=",
  print z
  z=range(1,7,2)
  print "range(1,7,2)=",
  print z

  x = 0.7071
  print "x = ",
  print x
  print "anyfunct(x) =    note complex result",
  print anyfunct(x)
  # end of main

def anyfunct(x): # thus main can be in front of needed functions, cmath also
  z = math.cos(x) + 1.0j*math.sin(x) - math.sqrt(x)
  return z


if __name__ == '__main__':
  main()

# end of hello.py see also  test_math.py

hello_py.out:
```

```
hello from python
hello from python
hello from python
a_int= 7
a_float= 7.25
a_real= 7.2597256357e+300
a_list= [7, 7.25, 'abc']
a_list[2]= abc
append= [7, 7.25, 'abc', 'def']
delete= [7, 'abc', 'def']
len(a_list)= 3

a_tuple= (2, 3, 5, 7)
b_tuple= (2, 3, 5, 7)
b_tuple[2]= 5

dict= {1: 'a', 2: 'b', 3: 'c'}
dict 5= {1: 'a', 2: 'b', 3: 'c', 5: 'e'}
dict[3]= c

range(5)= [0, 1, 2, 3, 4]
range(1,5)= [1, 2, 3, 4]
range(1,7,2)= [1, 3, 5]
x =  0.7071
anyfunct(x) =    note complex result (-0.080643380759+0.649631783705j)
```

loop, if then else, etc

Links to source file and output, shown below
for_else.py source file
for_else_py.out output file

```
# for_else.py  use of "else" after "for"
print "for_else.py running"
for i in range(0, 3): # : required  0,1,2  no 3
  print 'i= ', i
  for j in range(1,5,2): # step 2, nested loop 1, 3 no 5
    print 'i=', i, ' j= ', j

a=1
b=2
if(a==b or a!=b and a>b or a<b and a>=b or a<=b):
  print 'foo'
elif(a==a):      # any number of  elif
  print 'elif'
else:
  print 'bar'



for n in range(2, 21):
   for x in range(2, n):
       if n % x == 0:
           print n, 'equals', x, '*', n/x
           break
```

```
   else:
       # loop fell through without finding a factor
       print n, 'is a prime number'

# end for_else.py
```

```
for_else_py.out:
for_else.py running
i=  0
i= 0  j=  1
i= 0  j=  3
i=  1
i= 1  j=  1
i= 1  j=  3
i=  2
i= 2  j=  1
i= 2  j=  3
foo
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
20 equals 2 * 10
```

functions, procedures, internal and external

```
Note "import math" then  "print dir(math)" to see all available functions
```

```
Available math functions:
Links to source file and output, shown below
test_math.py  source file
test_math_py.out  output file
```

```
#test_math.py         sample use of functions after  import math
print "test_math.py running"
import math
print dir(math)                # ugly list of functions and values
print "math.pi=%g" % (math.pi)
print "math.e=%g" % (math.e)
print "math.sin(math.pi)=%10.3f" % (math.sin(math.pi))
print "math.cos(math.pi)=%10.3f" % (math.cos(math.pi))
print "math.tan(math.pi)=%g" % (math.tan(math.pi))
print "math.atan(1.0)=%10.3f" % (math.atan(1.0))
```

```
print "math.asin(1.0)=%10.3f" % (math.asin(1.0))
print "math.acos(0.5)=%10.3f" % (math.acos(0.5))
print "math.sinh(1.0)=%10.3f" % (math.sinh(1.0))
print "math.cosh(1.0)=%10.3f" % (math.cosh(1.0))
print "math.tanh(1.0)=%10.3f" % (math.tanh(1.0))
print "math.sqrt(2.0)=%10.3f" % (math.sqrt(2.0))
print "math.exp(2.0)=%10.3f" % (math.exp(2.0))
print "math.log(2.0)=%10.3f" % (math.log(2.0))
print "min(1.0,2.0)=%10.3f" % (min(1.0,2.0))
print "max(1.0,2.0)=%10.3f" % (max(1.0,2.0))
print "abs(-2.5)=%10.3f" % (abs(-2.5))
print "min(1,2)=%d" % (min(1,2))
print "max(1,2)=%d" % (max(1,2))
print "abs(-2)=%d" % (abs(-2))
a=complex(1.0,2.0)
print "a=complex(1.0,2.0)= %f %f" % (a.real,a.imag)
b=(3.0-2.0j)
print "b=(3.0-2.0j)= %f %f" % (b.real,b.imag)
c=a*b
print "c=a*b= (%f+ %fj)" % (c.real,c.imag)
print "no sqrt of complex"
print "math.sqrt(abs(c))=%10.3f" % (math.sqrt(abs(c)))
print "big number =%g" % (1.23456e300)
print "small number =%g" % (1.23456e-300)
print "end test_math.py"

test_math_py.out:
test_math.py running
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp',
'log', 'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh']
math.pi=3.14159
math.e=2.71828
math.sin(math.pi)=     0.000
math.cos(math.pi)=    -1.000
math.tan(math.pi)=-1.22461e-16
math.atan(1.0)=     0.785
math.asin(1.0)=     1.571
math.acos(0.5)=     1.047
math.sinh(1.0)=     1.175
math.cosh(1.0)=     1.543
math.tanh(1.0)=     0.762
math.sqrt(2.0)=     1.414
math.exp(2.0)=     7.389
math.log(2.0)=     0.693
math.pow(2.0,3)=     8.000
min(1.0,2.0)=     1.000
max(1.0,2.0)=     2.000
abs(-2.5)=     2.500
min(1,2)=1
max(1,2)=2
abs(-2)=2
a=complex(1.0,2.0)= 1.000000 2.000000
b=(3.0-2.0j)= 3.000000 -2.000000
c=a*b= (7.000000+ 4.000000j)
no sqrt of complex
math.sqrt(abs(c))=     2.839
big number =1.23456e+300
```

```
small number =1.23456e-300
end test_math.py
```

Links to source file and output, shown below
funct_to_import.py source file
test_import_funct.py source file
test_import_funct_py.out output file

```
# funct_to_import.py  just has  funct(n) that returns n
def funct(n):
  return n

a,b,c=0,0,0
def rabc():
  a=1
  b="rabc string"
  c=[3, 4, 5, 6]
  return a,b,c      #  note many values may be returned


# test_import_funct.py   import  funct_to_import and run funct and rabc
import funct_to_import
print  funct_to_import.funct(3)

print  funct_to_import.rabc()
x,y,z = funct_to_import.rabc()
print "z[1]=",
print z[1]
```

test_import_funct_py.out:

```
3
(1, 'rabc string', [3, 4, 5, 6])
z[1]= 4
```

Links to source file and output, shown below
test_passing_function.py source file
test_passing_function_py.out output file

```
# test_passing_function.py
def f(x):
  return x*x

def trap_int(f, xmin, xmax, nstep): # integrate f(x) from xmin to xmax
  area=(f(xmin)+f(xmax))/2.0
  h = (xmax-xmin)/nstep
  for i in range(1,nstep):
    x = xmin+i*h
    area = area + f(x)

  return area*h # trapezoidal method

print "test_passing_function.py running"
xmin = 1.0
xmax = 2.0
n = 10
area = trap_int(f, xmin, xmax, n)
print "trap_int area under x*x from ",xmin," to ",xmax," = ",area
```

```
test_passing_function_py.out:
test_passing_function.py running
trap_int area under x*x from  1.0  to  2.0  =  2.335
```

some sample plots

"import" needs a python library to be available.
sudo apt install python-tk

Links to source file .py and .out output file, click to read any not shown
plot_parab.py source file
plot_parab_py.jpg plot

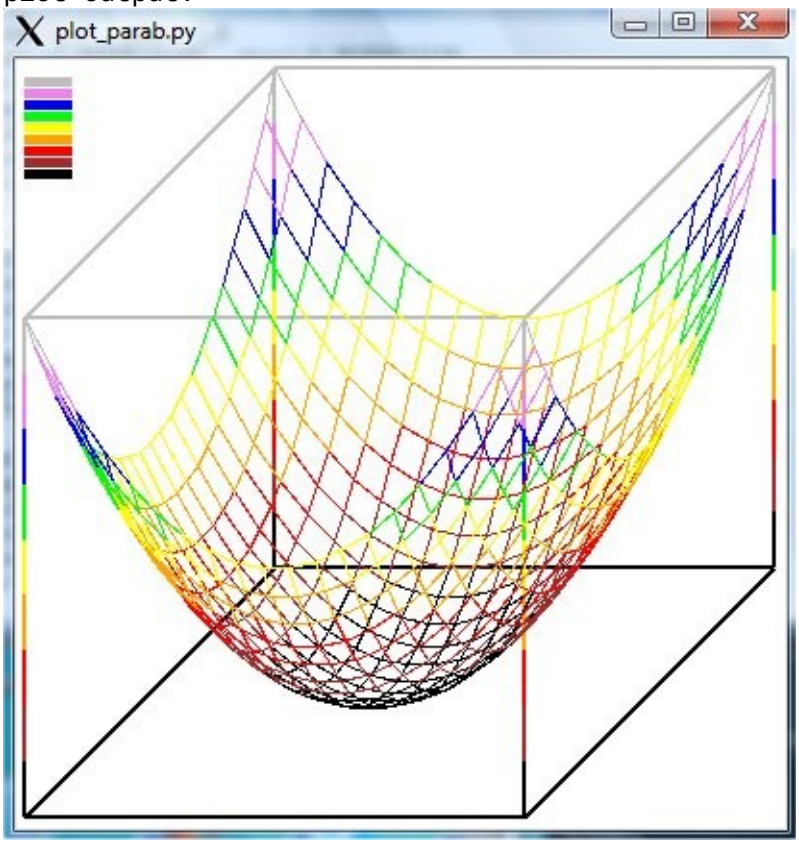
```
# plot_parab.py
from Tkinter import *

root=Tk()
root.title('plot_parab.py')
winw=400
winh=400
canvas=Canvas(root,width=winw,height=winh,bg='white')

nx=20
ny=25
xmin=-1.0
xmax=1.0
print 'xmin=',
print xmin,
print ', xmax=',
print xmax
ymin=-1.0
ymax=1.0
print 'ymin=',
print ymin,
print ', ymax=',
print ymax
zmin=1000.0
zmax=-1000.0
hx=(xmax-xmin)/(nx-1)
hy=(ymax-ymin)/(ny-1)
# for plotting
ya=0.5 # orthographic typ 0.1 to 0.5
zcol=['black', 'brown', 'red', 'orange', 'yellow', 'green', 'blue', 'violet',
'gray']
ncol=len(zcol)

for i in range(ncol):
  canvas.create_line(5,60-6*i,30,60-6*i,width=5,fill=zcol[i])

def parab(x,y):
  return x*x+y*y

def ortho(x, y, z, xm, ym, zm, xs, ys, ya, winh):
  x1=x-xm # zero base
  y1=y-ym
```

```
  z1=z-zm
  x2=x1+y1*ya # ortho z is up, y is back
  y2=z1+y1*ya
  x3=x2*xs # scale
  y3=y2*ys
  return x3+5, winh-y3-5 # away from boarder

zp=[[0 for j in range(ny)] for i in range(nx)]

for i in range(nx):
  x=xmin+hx*i
  for j in range(ny):
    y=ymin+hy*j
    z=parab(x,y)
    zp[i][j]=z
    zmin=min(z,zmin)
    zmax=max(z,zmax)

print 'zmin=',
print zmin,
print ', zmax=',
print zmax

# orthographic projection  scaled to winw, winh
xs=(xmax-xmin)
ys=(ymax-ymin)
zs=(zmax-zmin)
xs=xs+ys*ya
ys=zs+ys*ya
xs=(winw-10)/xs
ys=(winh-10)/ys

print 'ya=',
print ya,
print 'xs=',
print xs,
print ', ys=',
print ys

x1, y1 = ortho(xmin, ymin, zmin, xmin, ymin, zmin, xs, ys, ya, winh)
x2, y2 = ortho(xmax, ymin, zmin, xmin, ymin, zmin, xs, ys, ya, winh)
x3, y3 = ortho(xmax, ymax, zmin, xmin, ymin, zmin, xs, ys, ya, winh)
x4, y4 = ortho(xmin, ymax, zmin, xmin, ymin, zmin, xs, ys, ya, winh)
canvas.create_line(x1,y1,x2,y2,width=2,fill=zcol[0])
canvas.create_line(x2,y2,x3,y3,width=2,fill=zcol[0])
canvas.create_line(x3,y3,x4,y4,width=2,fill=zcol[0])
canvas.create_line(x4,y4,x1,y1,width=2,fill=zcol[0])

x5, y5 = ortho(xmin, ymin, zmax, xmin, ymin, zmin, xs, ys, ya, winh)
x6, y6 = ortho(xmax, ymin, zmax, xmin, ymin, zmin, xs, ys, ya, winh)
x7, y7 = ortho(xmax, ymax, zmax, xmin, ymin, zmin, xs, ys, ya, winh)
x8, y8 = ortho(xmin, ymax, zmax, xmin, ymin, zmin, xs, ys, ya, winh)
canvas.create_line(x5,y5,x6,y6,width=2,fill=zcol[ncol-1])
canvas.create_line(x6,y6,x7,y7,width=2,fill=zcol[ncol-1])
canvas.create_line(x7,y7,x8,y8,width=2,fill=zcol[ncol-1])
canvas.create_line(x8,y8,x5,y5,width=2,fill=zcol[ncol-1])

for k in range(ncol):
  dx=(x5-x1)/(ncol)
```

```
  xt=x1+k*dx
  xu=xt+dx
  dy=(y5-y1)/(ncol)
  yt=y1+k*dy
  yu=yt+dy
  canvas.create_line(xt,yt,xu,yu,width=2,fill=zcol[k])

for k in range(ncol):
  dx=(x6-x2)/(ncol)
  xt=x2+k*dx
  xu=xt+dx
  dy=(y6-y2)/(ncol)
  yt=y2+k*dy
  yu=yt+dy
  canvas.create_line(xt,yt,xu,yu,width=2,fill=zcol[k])

for k in range(ncol):
  dx=(x7-x3)/(ncol)
  xt=x3+k*dx
  xu=xt+dx
  dy=(y7-y3)/(ncol)
  yt=y3+k*dy
  yu=yt+dy
  canvas.create_line(xt,yt,xu,yu,width=2,fill=zcol[k])

for k in range(ncol):
  dx=(x8-x4)/(ncol)
  xt=x4+k*dx
  xu=xt+dx
  dy=(y8-y4)/(ncol)
  yt=y4+k*dy
  yu=yt+dy
  canvas.create_line(xt,yt,xu,yu,width=2,fill=zcol[k])


for i in range(nx-1):
  x=xmin+hx*i
  for j in range(ny-1):
    y=ymin+hy*j
    z1=zp[i][j]
    x1, y1 = ortho(x,    y, z1, xmin, ymin, zmin, xs, ys, ya, winh)
    z2=zp[i+1][j]
    x2, y2 = ortho(x+hx, y, z2, xmin, ymin, zmin, xs, ys, ya, winh)
    z3=zp[i][j+1]
    x3, y3 = ortho(x, y+hy, z3, xmin, ymin, zmin, xs, ys, ya, winh)
    icol=int((ncol/2.0)*(z1+z2)/(zmax-zmin))
    canvas.create_line(x1,y1,x2,y2,fill=zcol[icol])
    icol=int((ncol/2.0)*((z1-zmin)+(z3-zmin))/(zmax-zmin))
    canvas.create_line(x1,y1,x3,y3,fill=zcol[icol])

for i in range(nx-1):
  x=xmin+hx*i
  y=ymax
  z1=zp[i][ny-1]
  x1, y1 = ortho(x, y, z1, xmin, ymin, zmin, xs, ys, ya, winh)
  x=x+hx
  z2=zp[i+1][ny-1]
  x2, y2 = ortho(x, y, z2, xmin, ymin, zmin, xs, ys, ya, winh)
  icol=int((ncol/2.0)*((z1-zmin)+(z2-zmin))/(zmax-zmin))
```

```
  canvas.create_line(x1,y1,x2,y2,fill=zcol[icol])

for j in range(ny-1):
  y=ymin+hy*j
  x=xmax
  z1=zp[nx-1][j]
  x1, y1 = ortho(x, y, z1, xmin, ymin, zmin, xs, ys, ya, winh)
  y=y+hy
  z2=zp[nx-1][j+1]
  x2, y2 = ortho(x, y, z2, xmin, ymin, zmin, xs, ys, ya, winh)
  icol=int((ncol/2.0)*((z1-zmin)+(z2-zmin))/(zmax-zmin))
  canvas.create_line(x1,y1,x2,y2,fill=zcol[icol])

canvas.pack()
root.mainloop()
```

plot output:



Changes python2 to python3

```
I keep both compilers on my computer and have many source files of both.
Thus, I need to have a different command and different file extension.
e.g  python3 hello_world.py3
If you want to use both, you need a complete install of both because
each has a large library.
Typing  python -v  # shows version   mine says  python 2.7.5

The biggest change is "print" no longer a command, now a function:

Links to source file .py and .out output file, click to read any not shown
```

hello.py3 source file
hello_py3.out output file

```
# hello.py3  example file of hello.py in python2 changed to python3
print("hello from python")  # three ways to print the same line
print('hello from python')
print("hello", end=' ')
print("from python")
a_int=7
a_float=7.25
a_real=7.2597256357e+300
print("a_int=", a_int)
print("a_float=", a_float)
print("a_real=", a_real)

a_list=[7, 7.25, "abc"]
print("a_list=", a_list)
print("a_list[2]=", a_list[2])

a_tuple=2, 3, 5, 7
b_tuple=(2, 3, 5, 7) # same as above
print("a_tuple=", a_tuple)
print("b_tuple=", b_tuple)
print("b_tuple[2]=", b_tuple[2])

hello_py3.out:
hello from python
hello from python
hello from python
a_int= 7
a_float= 7.25
a_real= 7.2597256357e+300
a_list= [7, 7.25, 'abc']
a_list[2]= abc
a_tuple= (2, 3, 5, 7)
b_tuple= (2, 3, 5, 7)
b_tuple[2]= 5
```

Go to Top

Last updated 2/24/2017