Computational Physics

Python Programming Basics

Prof. Paul Eugenio
Department of Physics
Florida State University
Jan 17, 2019

http://hadron.physics.fsu.edu/~eugenio/comphy/

Announcements

Exercise 0 due end of day Friday

Arithmetic in Python

In most places where you can use a single variable in Python you can also use a mathematical expression

```
print(height + offset)
```

Basic mathematical operations

```
x + y          # addition
x - y          # subtraction
x * y          # multiplication
x / y          # division‡
x**y           # raising x to the power y
x // y         # integer division: example 5//3 returns 1
x % y          # integer modulo remainder: example 5%3 returns 2
```

‡ requires "`from __future__ import division`"

Easy Readable Expressions

Several mathematical operations can be combined together to make a more complicated expression along with the use of parentheses () to dictate the order of mathematical operations

```
length = (x-x0)*(x-x0) + (y-y0)**2 + (z-z0)**2
```

- Add spaces between the parts of a mathematical expression to make it easier to read.
- When different priorities are used, add a space around the operators with the lowest priority(ies). Never use more than one space, and always have the same amount of whitespace on both sides of a binary operator.

YES:

```
i = i + 1
i += 1
x = x*2 – 1
hypo2 = x*x + y*y
c = (a+b) * (a-b)
xSquared = x**2
```

NO:

```
i=i+1
i  +=1
x = x * 2 – 1
hypo2 = x * x+y * y
c = (a +b) * (a- b)
xSquared=x                **
```

Arithmetic

Tricks and Short Cuts

```
x += 1        # add 1 to x, same as x = x + 1
x -= 4        # subtract 4 from x, same as x = x - 4
x *= -2.6     # x = x * -2.6
x /= 5*y      # x = x / (5*y)
x //= 3.4     # x = int(x / 3.4)
```

Multiple Assignments in a Single Statement

```
x, y = 1, 2.5
x, y = 2*x*y+1, (x+y)/3
```

It is important to note that the whole right-hand side of the equation is calculated by the computer before assigning the left-hand values

Example: Multiple Assignments

Sequential Assignments

```
x = 1                          # value:  x = 1
y = 1                          # values: X = 1 & y = 1

x = 2*y + 1                    # values: x = 3 & y = 1
y = 2*x + 1                    # values: x = 3 & y = 7
```

Simultaneous Assignments

```
x, y = 1, 1                    # values: x = 1 & y = 1

x,y = 2*y + 1, 2*x +1          # values: x = 3 & y = 3
```

Be Careful What You Want May Not Be What You Get

Functions, Packages, and Modules

Python comes with facilities for performing many operations, from basic arithmetic to complex calculations, visualizations, and operations. These facilities are included in modules and packages(a collection of modules).

In order to use these facilities, one must import the function or module into the program.

Standard Mathematical Functions are provided in the math module.

```
log()                          # natural logarithm
log10()                        # log base 10
exp()                          # exponential
sin(), cos(), tan()            # trig functions(radians)
asin(), acos(), atan()         # inverse trig funct.
sinh(), cosh(), tanh()         # hyperbolic trig funct.
sqrt()                         # positive square root
```

plus values for π and *e* and many more functions

math – Mathematical functions

Using interactive python for help

```
hpc-login 434% python
Python 2.7.5
Type "help", "copyright", "credits" or "license" for more information.

>>> help("math")
Help on module math:

NAME
    math

FILE
    /opt/python27/anaconda/lib/python2.7/lib-dynload/math.so

MODULE DOCS
    http://docs.python.org/library/math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        Return the hyperbolic arc cosine (measured in radians) of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.
```

Importing Math Functions

Importing a Module

```
import math

math.sin(1.57)                              # returns 0.9999996..
math.sin(math.pi/2)                         # returns 1.0
math.sin(math.degrees(45))                  # returns 0.707107..
math.degrees(math.asin(1/math.sqrt(2)))     # equals 45
```

Importing Functions from a Module

```
from math import sin, asin, pi, degrees, sqrt

sin(1.57)                                   # returns 0.9999996..
sin(pi/2)                                   # returns 1.0
sin(degrees(45))                            # returns 0.707107..
degrees(asin(1/sqrt(2)))                    # equals 45
```

avoid using: "`from math import *`" (import all functions) as it can cause unknown behavior

Avoid import *

Import All Functions From a Module

```
from math import *              # a quick and dirty way to import

sin(1.57)
sin(pi/2)
```

Importing Functions over Functions

```
from math import *
from numpy import *         # NumPy is the fundamental package for
                            # scientific computing with Python.

sin(1.57)                               # which sin()?
sin(pi/2)                               #  math or numpy?
```

It's Better to Import Modules and Scope Functions

```
import math as m
import numpy as np

m.sin(1.57)
np.sin(1.57)
np.sin([np.pi,np.pi/2])   # returns: array([ 1.22464680e-16, 1.00e+00])
```

A ball dropped from a tower

A ball is dropped from a tower of height h with initial velocity zero. Write a program that asks the user to enter the height of the tower in meters and then calculates and prints the time the ball takes until it hits the ground, ignoring air resistance. Use your program to calculate the time for a ball dropped from a 100 m high tower.

A ball dropped from a tower

```
#! /usr/bin/env python

# ball_drop.py is program which calculates the time for a
# ball to drop from a height to the ground. It asks the
# user to enter a value for height (in meters) and prints
# the results to the screen.
#
# Paul Eugenio
# PHZ4151C
# Jan 17, 2019

# program header code
from __future__ import division, print_function
import math as m

#
# main body of program
#
g = 9.81    # acceleration due to gravity at the Earth's surface (m/s**2)

height = float(raw_input("Enter the height of the tower (in meters): "))

# distance = a*t**2/2
time = m.sqrt(2*height/g)

print("The ball takes", time, "seconds to hit the ground from a height of",
      height,"meters." )
```

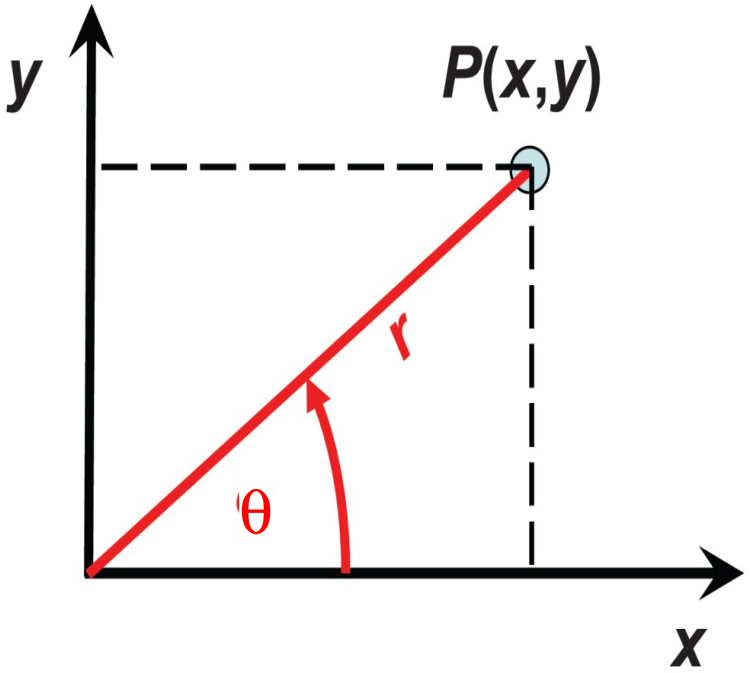
A ball dropped from a tower

A ball is dropped from a tower of height h with initial velocity zero. Write a program that asks the user to enter the height of the tower in meters and then calculates and prints the time the ball takes until it hits the ground, ignoring air resistance. Use your program to calculate the time for a ball dropped from 100 m high tower.

```
hpc-login 480% nedit ball_drop.py &

hpc-login 480% chmod +x ball_drop.py

hpc-login 481% ball_drop.py          ( or ./ball_drop.py )

Enter the height of the tower (in meters): 100
The ball takes 4.515 seconds to hit the ground from a height of 100.0 meters.

hpc-login 482%
```

Polar to Cartesian Coordinates

Write a program to perform the inverse operation to that of Example 2.2. That is, ask the user for the Cartesian coordinates x, y of a point in two-dimensional space, and calculate and print the corresponding polar coordinates, with the angle θ given in degrees.



$x = r \cos(\theta)$

$y = r \sin(\theta)$

$r = \sqrt{x*x + y*y}$

$\theta = \text{atan}(y/x)$

The atan() math function is limited to +90° to -90°

see python: help("math")

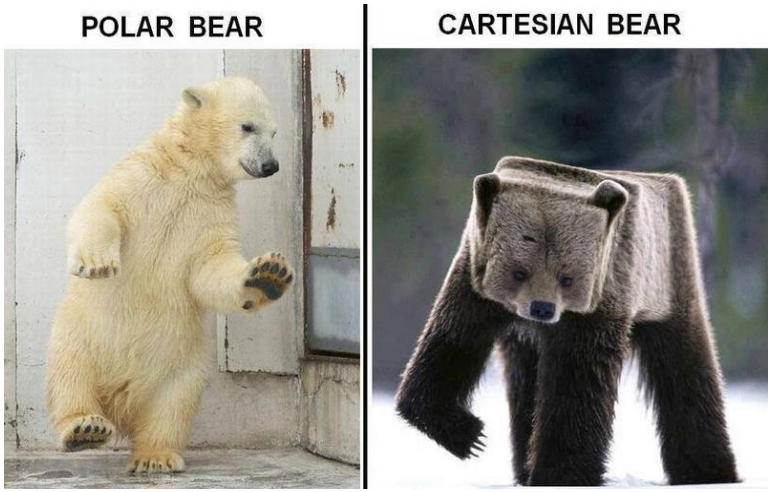
```
atan2(...)
        atan2(y, x)

        Return the arc tangent (measured in radians) of y/x.
        Unlike atan(y/x), the signs of both x and y are considered.
```

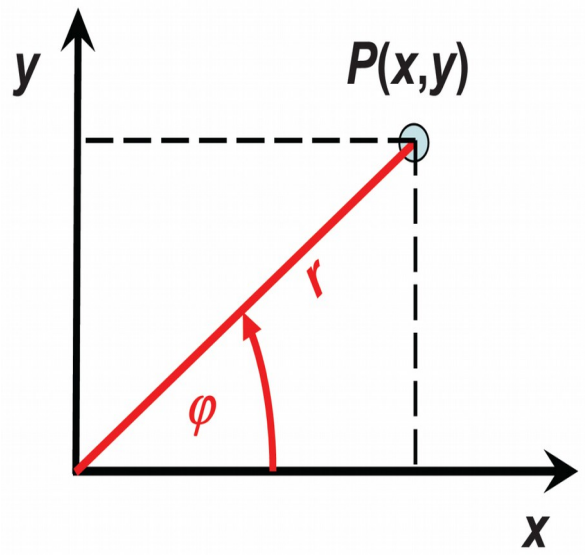
Polar to Cartesian Coordinates

```
#! /usr/bin/env python

# cartesian2polar.py is program which calculates the 2D radius and
# polar angle given the cartesian coordinates.  The results are printed
# to the screen.
#
# Paul Eugenio
# PHZ4151C
# Jan 17, 2019

# program header code
from __future__ import division, print_function
import math as m

# main body of program

# get input values
x = float(raw_input("Enter the x coordinate: "))
y = float(raw_input("Enter the y coordinate: "))

# transform values from cartesian to polar
r = m.sqrt(x*x + y*y)
phi = m.atan2(y, x)


print("The polar radius is", r, ", and the polar angle is",
      m.degrees(phi), "degrees.")
```

Polar to Cartesian Coordinates

Write a program to perform the inverse operation to that of Example 2.2. That is, ask the user for the Cartesian coordinates x, y of a point in two-dimensional space, and calculate and print the corresponding polar coordinates, with the angle θ given in degrees.






```
hpc-login 480% nedit cartesian2polar.py &

hpc-login 480% chmod +x cartesian2polar.py

hpc-login 481% cartesian2polar.py
Enter the x coordinate: 2
Enter the y coordinate: 2
The polar radius is 2.82842712475 and the polar angle is 45.0 degrees.

hpc-login 482%
```

This Week's Exercise

Python Basics

Exercise set #1

DUE DATE: January 25

Let's get working