

# PyOhio PyCamp™ 2014

## The Original Python Boot Camp Meets the 7th Annual Ohio Regional Python Conference

The free PyOhio conference has teamed up again with PyCamp, the ultra-low-cost Python Boot Camp for beginners. PyCamp makes you productive so you can get your work done quickly. Python is used in a variety of applications from bioinformatics to geographic information systems to motion picture production. PyCamp emphasizes the features which make Python a simpler and more efficient language. Following along with example Python PushUps speeds your learning process in a modern high-tech classroom. Register today for training in the programming language used by Google and NASA. Learn to create your own Python modules in just five days. PyCamp is conducted on the campus of Ohio State University in the Ohio Union during the week leading up to the PyOhio weekend conference.



<http://tripython.org/pyohio14>  
July 21-25, 2014

Ohio Union  
Ohio State University

A program of PyOhio: <http://pyohio.org>



```
import turtle, random
import math
class Turtle:
    def __init__(self, color):
        self.__color = color
        self.__x = 0
        self.__y = 0
        self.__heading = 0
    def move(self, distance):
        self.__x = self.__x + distance * math.cos(self.__heading)
        self.__y = self.__y + distance * math.sin(self.__heading)
    def turn(self, amount):
        self.__heading = (self.__heading + amount) % 360
    def draw(self, distance):
        turtle.penup()
        turtle.goto(self.__x, self.__y)
        turtle.pendown()
        turtle.forward(distance)
    def __str__(self):
        return "Turtle at (%s, %s) heading %s" % (self.__x, self.__y, self.__heading)
    def __repr__(self):
        return "Turtle(%s, %s, %s)" % (self.__x, self.__y, self.__heading)
    def __del__(self):
        pass
    def __eq__(self, other):
        return self.__x == other.__x and self.__y == other.__y and self.__heading == other.__heading
    def __neq__(self, other):
        return not self.__eq__(other)
    def __lt__(self, other):
        return self.__heading < other.__heading
    def __gt__(self, other):
        return self.__heading > other.__heading
    def __le__(self, other):
        return self.__heading <= other.__heading
    def __ge__(self, other):
        return self.__heading >= other.__heading
    def __add__(self, other):
        return Turtle(self.__x + other.__x, self.__y + other.__y, self.__heading + other.__heading)
    def __sub__(self, other):
        return Turtle(self.__x - other.__x, self.__y - other.__y, self.__heading - other.__heading)
    def __mul__(self, other):
        return Turtle(self.__x * other.__x, self.__y * other.__y, self.__heading * other.__heading)
    def __div__(self, other):
        return Turtle(self.__x / other.__x, self.__y / other.__y, self.__heading / other.__heading)
    def __mod__(self, other):
        return Turtle(self.__x % other.__x, self.__y % other.__y, self.__heading % other.__heading)
    def __and__(self, other):
        return Turtle(self.__x & other.__x, self.__y & other.__y, self.__heading & other.__heading)
    def __or__(self, other):
        return Turtle(self.__x | other.__x, self.__y | other.__y, self.__heading | other.__heading)
    def __xor__(self, other):
        return Turtle(self.__x ^ other.__x, self.__y ^ other.__y, self.__heading ^ other.__heading)
    def __radd__(self, other):
        return Turtle(self.__x + other, self.__y + other, self.__heading + other)
    def __rsub__(self, other):
        return Turtle(self.__x - other, self.__y - other, self.__heading - other)
    def __rmul__(self, other):
        return Turtle(self.__x * other, self.__y * other, self.__heading * other)
    def __rdiv__(self, other):
        return Turtle(self.__x / other, self.__y / other, self.__heading / other)
    def __rmod__(self, other):
        return Turtle(self.__x % other, self.__y % other, self.__heading % other)
    def __rand__(self, other):
        return Turtle(self.__x & other, self.__y & other, self.__heading & other)
    def __ror__(self, other):
        return Turtle(self.__x | other, self.__y | other, self.__heading | other)
    def __rxor__(self, other):
        return Turtle(self.__x ^ other, self.__y ^ other, self.__heading ^ other)
    def __rdivmod__(self, other):
        return Turtle(self.__x // other, self.__y // other, self.__heading // other)
    def __divmod__(self, other):
        return Turtle(self.__x // other, self.__y // other, self.__heading // other)
    def __pow__(self, other):
        return Turtle(self.__x ** other, self.__y ** other, self.__heading ** other)
    def __rpow__(self, other):
        return Turtle(self.__x ** other, self.__y ** other, self.__heading ** other)
    def __getitem__(self, index):
        return self.__x, self.__y, self.__heading
    def __setitem__(self, index, value):
        self.__x, self.__y, self.__heading = value
    def __iter__(self):
        return iter([self.__x, self.__y, self.__heading])
    def __len__(self):
        return 3
    def __contains__(self, item):
        return item in [self.__x, self.__y, self.__heading]
    def __delitem__(self, index):
        del self.__x, self.__y, self.__heading
    def __copy__(self):
        return Turtle(self.__x, self.__y, self.__heading)
    def __deepcopy__(self, memo):
        return Turtle(self.__x, self.__y, self.__heading)
    def __hash__(self):
        return hash((self.__x, self.__y, self.__heading))
    def __str__(self):
        return "Turtle at (%s, %s) heading %s" % (self.__x, self.__y, self.__heading)
    def __repr__(self):
        return "Turtle(%s, %s, %s)" % (self.__x, self.__y, self.__heading)
    def __del__(self):
        pass
    def __eq__(self, other):
        return self.__x == other.__x and self.__y == other.__y and self.__heading == other.__heading
    def __neq__(self, other):
        return not self.__eq__(other)
    def __lt__(self, other):
        return self.__heading < other.__heading
    def __gt__(self, other):
        return self.__heading > other.__heading
    def __le__(self, other):
        return self.__heading <= other.__heading
    def __ge__(self, other):
        return self.__heading >= other.__heading
    def __add__(self, other):
        return Turtle(self.__x + other.__x, self.__y + other.__y, self.__heading + other.__heading)
    def __sub__(self, other):
        return Turtle(self.__x - other.__x, self.__y - other.__y, self.__heading - other.__heading)
    def __mul__(self, other):
        return Turtle(self.__x * other.__x, self.__y * other.__y, self.__heading * other.__heading)
    def __div__(self, other):
        return Turtle(self.__x / other.__x, self.__y / other.__y, self.__heading / other.__heading)
    def __mod__(self, other):
        return Turtle(self.__x % other.__x, self.__y % other.__y, self.__heading % other.__heading)
    def __and__(self, other):
        return Turtle(self.__x & other.__x, self.__y & other.__y, self.__heading & other.__heading)
    def __or__(self, other):
        return Turtle(self.__x | other.__x, self.__y | other.__y, self.__heading | other.__heading)
    def __xor__(self, other):
        return Turtle(self.__x ^ other.__x, self.__y ^ other.__y, self.__heading ^ other.__heading)
    def __radd__(self, other):
        return Turtle(self.__x + other, self.__y + other, self.__heading + other)
    def __rsub__(self, other):
        return Turtle(self.__x - other, self.__y - other, self.__heading - other)
    def __rmul__(self, other):
        return Turtle(self.__x * other, self.__y * other, self.__heading * other)
    def __rdiv__(self, other):
        return Turtle(self.__x / other, self.__y / other, self.__heading / other)
    def __rmod__(self, other):
        return Turtle(self.__x % other, self.__y % other, self.__heading % other)
    def __rand__(self, other):
        return Turtle(self.__x & other, self.__y & other, self.__heading & other)
    def __ror__(self, other):
        return Turtle(self.__x | other, self.__y | other, self.__heading | other)
    def __rxor__(self, other):
        return Turtle(self.__x ^ other, self.__y ^ other, self.__heading ^ other)
    def __divmod__(self, other):
        return Turtle(self.__x // other, self.__y // other, self.__heading // other)
    def __rdivmod__(self, other):
        return Turtle(self.__x // other, self.__y // other, self.__heading // other)
    def __pow__(self, other):
        return Turtle(self.__x ** other, self.__y ** other, self.__heading ** other)
    def __rpow__(self, other):
        return Turtle(self.__x ** other, self.__y ** other, self.__heading ** other)
    def __getitem__(self, index):
        return self.__x, self.__y, self.__heading
    def __setitem__(self, index, value):
        self.__x, self.__y, self.__heading = value
    def __iter__(self):
        return iter([self.__x, self.__y, self.__heading])
    def __len__(self):
        return 3
    def __contains__(self, item):
        return item in [self.__x, self.__y, self.__heading]
    def __delitem__(self, index):
        del self.__x, self.__y, self.__heading
    def __copy__(self):
        return Turtle(self.__x, self.__y, self.__heading)
    def __deepcopy__(self, memo):
        return Turtle(self.__x, self.__y, self.__heading)
    def __hash__(self):
        return hash((self.__x, self.__y, self.__heading))
```