classes and objects

http://www.youtube.com/watch?v=L57-vQvo34E

python™

OOP in python

- python was built as a *procedural language*
- as a result, python's OOP feels tacked-on when compared to java

defining a class

```
class Name:
     statements
```

- class name is capitalized
- saved into file `name.py` (lowercase)

fields

```
class Point:
    x = 0
    y = 0
```

using a class

```
from point import *

p1 = Point()
p1.x = 7
p1.y = -3
```

- import file name (lowercase), not class name
- no new operator like in java

methods

```python
def translate(self, dx, dy):
    self.x += dx
    self.y += dy
```

- first parameter must be `self`
    - use `self.name` to access fields

exercise

- write the following methods for `Point`:
  - `set_location`
  - `distance`
  - `draw`

point.py

```python
from math import *

class Point:
    x = 0
    y = 0

    def set_location(self, x, y):
        self.x = x
        self.y = y

    def distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx * dx + dy * dy)

    def draw(self, panel, color="black"):
        panel.canvas.create_oval(self.x, self.y,
            self.x + 3, self.y + 3,
            fill=color, outline=color)
        panel.canvas.create_text(self.x, self.y,
            text=str(self), anchor="sw", fill=color)
```

constructors

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

- special method named `__init__` called when creating an object
- now we can create points by saying `p = Point(5, 23)`

declaring fields

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

- if fields are declared in the constructor, they don’t need to be declared outside it

printing objects

```
>>> p = Point(5, -2)
>>> print p
<Point instance at 0x00A8A850>
>>> str(p)
'<Point instance at 0x00A8A850>'
```

- ick. let's fix this.
- would write a `toString` method in java...

__str__

```
def __str__(self):
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```

- special method, automatically called when using `str` or `print`

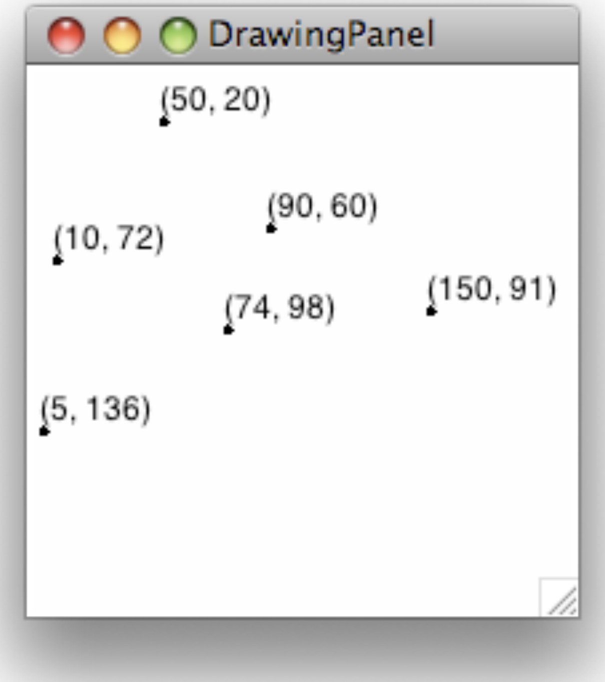
```
>>> p = Point(5, -2)
>>> print p
(5, -2)
>>> str(p)
'(5, -2)'
```

exercise!

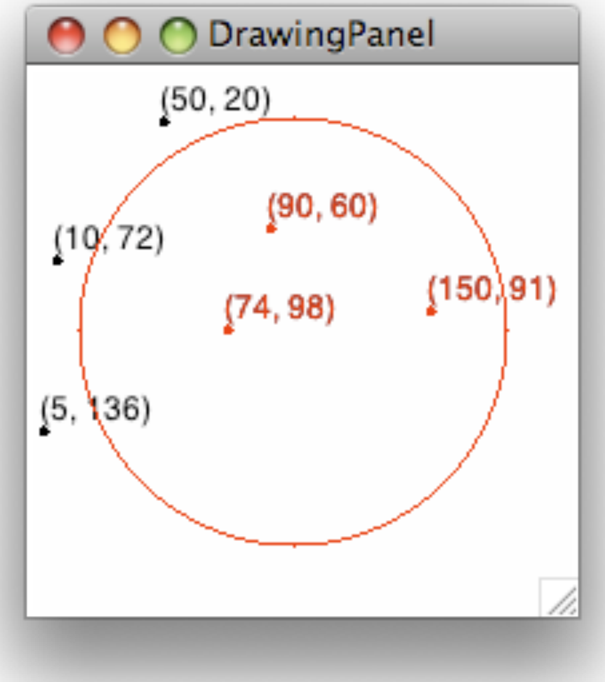
towns.txt

```
50  20
90  60
10  72
74  98
5   136
150 91
```

bomb.py



```
Blast site x? 100
Blast site y? 100
Blast radius? 80
```



- for an extra challenge, have it randomly choose the location of a town instead of prompting the user

bonus content!

higher-order functions

```
>>> def double(x):
...     return x + x
...
>>> def do_twice(func, x):
...     return func(func(x))
...
>>> do_twice(double, 5)
20
```

- in python, functions can be passed around as parameters

map

```
>>> map(double, [1, 2, 3, 10])
[2, 4, 6, 20]
```

- map(func, sequence) returns func(element) for every element in sequence

filter

```
>>> def is_odd(n):
...     return n % 2 == 1
...
>>> filter(is_odd, [1, 2, 3, 10])
[1, 3]
```

- filter(func, sequence) returns a list of every element in sequence for which func(element) is True

list comprehensions

```
>>> map(double, filter(is_odd, [1, 2, 3, 10]))
[2, 6]

>>> [n * 2 for n in [1, 2, 3, 10] if n % 2 == 1]
[2, 6]
```

- a concise way to `map`/`filter` without defining functions

exercise

- write the following functions:
  - `factors`(n) returns a list of factors of n
  - `is_prime`(n) returns `True` if n is prime
- produce a list of all primes less than 250

solution

```
def factors(n):
    return [m for m in range(1, n+1) if n % m == 0]

def is_prime(n):
    return factors(n) == [1, n]


>>> filter(is_prime, range(250))
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241]
```