Comparables

March 13, 2021

0.1 Obtaining the Boolean Representation

[6]:
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f'<Point(x={self.x}, y={self.y})>'

    def __bool__(self):
        return bool(self.y / self.y) if self.x != 0 else bool(self.y / (self.x␣
 ↪+ 1))

p1 = Point(0, 0)
p2 = Point(1, 3)

print(p1)
print(p2)
print("")
print("bool(p1)")
print(bool(p1))
print("bool(p2)")
print(bool(p2))
```

```
<Point(x=0, y=0)>
<Point(x=1, y=3)>

bool(p1)
False
bool(p2)
True
```

0.2 Equality/Inequality and Greater/Less

- in python `eq` and `ne` are the comparison operators

[11]:
```
from operator import eq
from operator import ne
from operator import ge, lt, le, gt

x = 1
y = 2

eq(x, y)
```

[11]:
```
False
```

- when comparing objects, you are comparing memory adresses

[21]:
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f'<Point(x={self.x}, y={self.y})>'

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __ne__(self, other):
        return not self.__eq__(other)
        #return self.x != other.x or self.y != other.y

    def __lt__(self, other):
        return self.x < other.x and self.y < other.y

    def __gt__(self, other):
        return self.x > other.x and self.y > other.y

    def __le__(self, other):
        return self.x <= other.x and self.y <= other.y

    def __ge__(self, other):
        return self.x >= other.x and self.y >= other.y

p1 = Point(0, 0)
p2 = Point(0, 0)
p3 = Point(1, -4)

print('p1 == p2')
print(p1 == p2)
print("")
print("p1 < p2")
```

```
print(p1 < p2)
print("")
print("p1 <= p2")
print(p1 <= p2)
```

```
p1 == p2
True

p1 < p2
False

p1 <= p2
True
```

0.3 Hashing and Slots

- dictionary keys have to be hashable inorder to have fast lookups
- implementing `__eq__` makes `__hash__ = None`
- meaning using `__eq__` makes you lose ability to hash
- `__slots__` property makes it so that an object can only have properties that you initialize it with
  - `__slots__ = ('size', 'color')`
  - how many slots are you getting per property
  - normally objects are dynamic because there is a backing dictionary to every object that can grow in size as it needs to
  - slots says you get two spaces
- you have to be careful if you use these as key in a dictionary, because they will not be garbage collected

[26]:
```
from random import randint

class Bacteria:

    __slots__ = ('color', 'size', 'id')

    def __init__(self, size, color):
        self.color = color
        self.size = size
        self.id = randint(0, 10000)

    def __hash__(self):
        # removes ability to have different object with same hash
        return hash((self.size, self.color, self.id))

    # if two things are equal, it assumes their hashes are equal
    def __eq__(self, other):
        return (
            self.__class__ == other.__class__ and
```

```
            self.size == other.size and
            self.color == other.color and
            self.id == other.id
        )

b1 = Bacteria(5, 'red')
b2 = Bacteria(10, 'blue')

reviews = {}
reviews[b1] = 'small but fast'
reviews[b2] = 'bigh but slow'

print("b1")
print(b1)
print("")
print("hash(b1)")
print(hash(b1))
print("")
print("hash(b2)")
print(hash(b2))
```

```
b1
<__main__.Bacteria object at 0x0000015A5BFBCEC0>

hash(b1)
5611925501173293849

hash(b2)
-1871468954426514467
```